UNIVERSITY OF OSLO
Department of Informatics

Dynamic Cloud Infrastructure

Espen Gundersen

Network and System Administration

University of Oslo

May 23, 2012

Dynamic Cloud Infrastructure

Espen Gundersen

Network and System Administration
University of Oslo

May 23, 2012

Abstract

This thesis will explore and investigate the possibility of implementing nested clouds to increase flexibility. A nested cloud is a private cloud running inside another private or public cloud.

The goal is to enable live migration of virtual machines across cloud boundaries. A virtual machine running in a cloud on top of a hypervisor could not traditionally be migrated to a different cloud using a different hypervisor. This creates boundaries and lock-in situations with loss of flexibility. By nesting a cloud it can span multiple different clouds and hypervisors to create a virtual private cloud. The various underlying platforms will then act similar to different physical server nodes in a traditional cloud.

An implementation using nested clouds was suggested and tested as an evolution of the private hybrid cloud. The working implementation could be a solution to the increasing importance of cloud independence.

For the nested cloud to be feasible it is required that the overhead by having virtualization on two layers is kept at a minimum. Throughout the thesis the performance was measured and analysed to maintain a high performing system, and the behaviour was observed to ensure robustness.

Acknowledgements

First and foremost I would like to express my appreciation of my supervisor, Ismail Hassan. I am very thankful for his guidance and genuine interest in my research and work throughout the project. The numerous discussions we have had has been a great inspiration and truly improved the thesis. His enthusiasm for the specific topic and field in general has confirmed its relevance and importance, and nurtured my motivation.

I will also thank my family and friends for the support. Their contributions and patience was highly appreciated and needed. I would like to extend a special thanks to my father Atle, and my friend and fellow student Eirik for their review of my work, and endless hours of discussion and technical brainstorm; and the occasional beer.

I am also grateful to all my classmates and lectures in the program and over the years for the heated discussions and motivation to pursue the masters degree. I also wish to draw attention to Kyrre M Begnum, as he has been greatly influential for my work over the years and a true source of inspiration.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Computer systems and data centers are essential in most businesses and for the wide private population. They are growing in size in line with the demand, and are becoming an increasing post in budgets and a source of concern for system administrators as management complexity increases. For private companies the cost of these data centers in form of power for cooling and operation of servers as well as maintaining and acquiring the servers itself are generally high[28][18][25]. To minimize the cost it is desirable to keep the number of servers down and optimize the resources as good as possible.

Virtualization of servers have been used to optimize resources over many years and is widely adopted in the industry today[31][12]. Virtualization also gives more flexibility and control for the administrators as virtual machines are far easier to manage in large scales. Through virtualization a single server could host multiple operating systems and thereby maximizing the resources and reducing the total number of physical servers.

Cloud computing has recently seen a growing adoption as an evolution of virtualization. Cloud is a more flexible and scalable solution to organize virtual machines over a data center. Although cloud computing started out as an on-demand, off-site resource-rental service hosted by large international corporations like Amazon it has evolved into a rich field of products and solutions. Due to security concerns and privacy issues private clouds have grown in popularity and offers many of the same advantages in maximum optimization of resources and cost control, but by being hosted in-house avoids many privacy concerns. Private clouds gives more control over security but the on-demand nature is reduced or lost due to ownership of hardware and operating costs. Ideally we want total control over the infrastructure and hardware, but retain the flexibility and on-demand ability by only paying for what is used. Traditionally these are contradictory requirements.

Additionally the diversity of solutions introduces the issues of portability and

compatibility. When implementing business applications on a public cloud platform it is highly likely that the system is dependent on functionality and APIs provided by the cloud technology. And without privileged access to the underlying system it is unlikely to be compatible with other systems by other cloud providers. The providers have data centers around the world and with a large variety of advanced features that require cloud specific implementations by the customer. It could be seen as advantageous for the provider to deliberately break the capability with the competitors. This is called *vendor lock-in*[3]. It prevents the customers from freely moving their business logic between providers or technologies.

But even when controlling all aspects of the cloud infrastructure this incompatibility is still present. Most private cloud platforms have unique solutions and APIs which hinders the portability between them. This becomes a problem if a decision to change platform is taken, but more importantly it eliminates any possibility of combining different platforms in a production environment. Moving an application system from one private cloud platform to another without rewriting and reimplementing the logic is not feasible with todays systems.
Some effort has been put into uniting the APIs for basic cloud operations. The Deltacloud[16][20] project and Open Cloud Computing Interface (OCCI)[11] aim to deliver a common interface to all major cloud systems. But this is still just for for example starting and stopping virtual machines and does not deliver the full flexibility to freely live migrate systems onto different platforms.

Throughout this paper I will look at what possibilities exists to increase the flexibility of private clouds, and how a truly platform independent solution that enables migration between any virtual infrastructure like private clouds can be created.

The concept will allow for a virtual system to be hosted on any other virtual infrastructure. This means that the virtual machines can be located on any standard consumer desktop or workstation, or in a private cloud data center with any platform of choice, or on a large scale public cloud. And the whole virtual infrastructure can be freely live migrated between them all based on the needs and policies.

The focus of this thesis will be on the hosting of fully hardware accelerated virtual machines. The hosting of these virtual machines are what cloud providers often label *compute*. For the complete concept to be implemented seamlessly it also depends on storage and networking, however this is outside the scope of this thesis.

1.2 Problem Statement

- How can flexibility and portability of a cloud be increased with nested virtualization?
- Is nesting of clouds feasible while maintaining viable performance?
- How could a whole cloud infrastructure be live migrated to a different platform without interruption?

The first question is what benefits that can be achieved by nesting a cloud. Increased flexibility means that the freedom to freely configure the cloud to the users requirements, even when hosted outside an environment, is under the administrators control. Portability is the ability to migrate virtual machines and infrastructure from one platform to another circumventing the vendor lock-ins and lack of standardization.

The second statement concerns the performance of such nested clouds. As all layers of virtualization involves an overhead it is crucial that the loss is minimal for the concept to remain attractive.

The third statement is about the ability to live migrate clouds across platform borders. A nested cloud could potentially live migrate between private clouds and environments and to a public cloud. This ability would increase flexibility and portability and further break any vendor lock-ins.

1.3 Approach

Enabling virtual machines to be migrated to a different platform without controlling the underlying infrastructure require a new layer of abstraction. This layer will interact with all the different platforms underneath and provide the same interface as above for the virtual machines to interact with. This way the virtual machines moving from platform to platform will always interact with the same virtual platform independent of the system at the bottom. For this theory to work two layers to virtualization, called *nested virtualization*, is needed.

The first area of research will be to explore the possibilities to run the cloud-nodes virtualized with nested virtualization. This means that the instances started by the customer will execute inside a virtual machine. The paper will look at how this could be done and to what extent it will be with satisfactory performance. The overhead of a single virtual machine must be measured, and then the additional overhead for the nested virtual machine. The loss of performance must be minimal for this to be a practical solution to gain flexibility. For nested hardware virtualization to work it must be supported in the hypervisor. If this is not the case, the second, virtual, hypervisor will be forced to fall back to emulation, or a paravirtual second hypervisor could be used. When

the performance overhead is measured and compared a conclusion whether nested clouds are feasible or not could be drawn.

Next a private cloud technology must be implemented in the virtualized environment and tested that it will maintain availability and connectivity while live migrating individual virtual machines. There are two virtual layers machines can be migrated onto, and both will be explored and tested.
The thesis will also explore and explain the combinations of nested clouds and where live migration can be achieved. The requirements and limitations of nested virtual clouds will be discussed and what possibilities the concept can open for.

1.4 Thesis Outline

The thesis is divided into six chapters:

The first part is an introduction to the problem and an overview of the approach. The current situation is described and leads to the motivation behind the thesis.

The second chapter is a summation of background research and technology used throughout this thesis.

The third part is where the setup and experiments are described. Firstly the concept is explained and the theory of the proposed solution clarified. Then the setup and environment used for implementation, and lastly chapter two describes the tests.

The fourth chapter display the results of the testing and presents the findings.

Chapter five is the analyses and discussion. Here the results are explained and their significance documented with statistical analysis. The findings and behavior of the implemented concept is discussed and reviewed from a flexibility, performance, security and privacy standpoint. Future work are suggested.

The last chapter concludes the thesis and summarizes the work.

Chapter 2

Background and literature

This chapter will introduce the background literature and previous relevant research. It will also cover the most significant technologies used throughout the thesis.

2.1 Virtualization

Virtualization is the term for something that is virtual, as opposed to real, but giving the impression of being actual. In computing this can come in many forms. In early years virtual memory was introduced in multitasking kernels to give a process the impression of having control over all the memory. This was an abstraction from the actual hardware so each program didn't have to coordinate with all other running proses to share the same memory. The kernel would provide each process with virtual memory so it would appear for that process to be alone. This technique is still the way to manage memory in multitasking operating systems.

Another form of virtualization is program execution environment. This is a way to execute a program inside a virtual environment. The best known is Javas *Java virtual machine* JVM, a virtual environment for executing Java bytecode. This enables great portability between systems since the program itself is only interacting with the JVM and not the underlying hardware. As long as the JVM supports the platform in question the program will run. The programming is abstracted from the actual hardware and simplified towards a generic platform.

The first occurrences of fully virtualized guest operating systems was seen with IBM in the late sixties and seventies. This was large mainframe systems with single-user guest operating systems. The x86 architecture did not support virtualization and was therefore not used as a platform for VMs.

In more recent years the abstraction has gone further by enabling virtualization on the x86 architecture. This results in the ability to abstract the whole kernel and operating system. A small layer of logic, called a hypervisor, sits

between the hardware and the system and provides a virtualized set of components to guests. A guest, often called a domain, is an operating system running on top of this layer, hypervisor. This hypervisor can host multiple guests and provide them with different components. The guests will then share the actual hardware and can run independent of each other providing there are sufficient resources. For the guests the components provided by the hypervisor looks and acts as actual hardware, but is just a virtualized abstraction.

2.1.1 Types of virtualization

The lack of hardware virtualization in the x86 architecture was for long a problem and hindered efficient virtualization. This left users of the x86 platform to emulation. Emulation is when the operating system translates instructions from one architecture to a different one. This means that every instruction is handled and require CPU time for the translation, which results in dramatic loss of performance.

Since the demand for virtualization on the x86 platform was present more software solutions where developed. The first significant method was *Dynamic binary translation*. DBT is similar to emulation in the sense that the instructions are translated on the fly, but opposed to emulation DBT translates instructions to the same architecture as the original. This enables the hypervisor to pass user-mode code untouched through without translation. Only kernel-mode code needs to be handled. This technique is so effective that the virtualized guests are close to the performance of code executing on the native machine. But even with this optimized translation there is still significant overhead and the IO devices still emulated and performed poorly.

The next breakthrough for x86 virtualization came with the introduction of Xen in 2003. Xen uses a technique called *paravirtualization*. It is still software virtualization as DBT, but with significantly less overhead due to less code executing in kernel-mode, the Achilles heel of DBT. Paravirtualization requires the guest to be aware of the hypervisor. This means that the guest know it is a virtual machine and communicates with the hypervisor. The negative side of this is that the operating system needs to be altered and adapted to use the features. This will break compatibility with most existing systems. On the positive side there is a performance gain due to less overhead and CPU time needed for translation by the hypervisor. Similar to DBT most user-mode code will execute directly without intervention by the hypervisor. But since the guest system now can communicate directly with the hypervisor (called hypercalls) it can limit the instructions that will trigger a translation and generate unnecessary overhead. This will also results in a very lightweight hypervisor. Another advantage is the performance improvement in IO devices. Xen will always require a *Dom0* guest, a privileged guest, will full access to all physical devices. The normal guests will then use special paravirtualized drivers to interact with them.

With the more recent virtualization modules in the Intel and AMD processors it is possible to utilize full hardware accelerated virtualization. This capability allows the system to enter guest mode and operations will not be interpreted by the hypervisor. This results in faster execution compared to paravirtualization by the switch from host to guest mode, since it has an overhead and use some CPU cycles. It is therefore faster when executing large operations with few switches, but with smaller operations and more switching the software virtualization are often faster.

The hardware virtualization support from Intel and AMD came in two stages. The first generation was purely for processing and did not include the memory. The second generation included memory management. Currently the production software is on that second generation, the third is coming and will include IO-devices such as network and disk controllers.

While waiting for the third generation of hardware virtualization for the x86 architecture it is possible to combine two types of virtualization to achieve the best performance. This means that the processor and memory are fully virtualized, but the IO devices still uses paravirtualization to improve performance.

The different hypervisors can also be categorized by how they are installed and implemented. Type 1 or often called bare-metal hypervisor, and resides directly on top of the hardware without any operating system in itself. While type 2, or hosted hypervisor, is a hypervisor hosted inside a normal operating system. Type 1 is the traditional architecture used by the old mainframe systems, Xen and VMware. The best example of a type 2 hypervisor is VirtualBox. This is a clear software installed inside an existing environment. Whether KVM is type 1 or 2 is more unclear and under discussion. Traditionally KVM would fall under type 2 since it is dependent on an existing operating system. But KVM is an integrated part of the Linux kernel and uses many of its functions directly, such as the scheduler and memory management. It is clearly software installed on top of a system, but it is also integrated so it could be argued that the Linux kernel itself is KVM and therefore a type 1.

2.1.2 Virtualization products

There are many different hypervisors currently on the market. Xen, KVM, VMWare and Hyper-V are the most significant, and apart from some features and characteristics whey provide the same basic service.
Of these four Xen and KVM are licenced under the open source licence GPL. Xen is owned by Citrix and KVM by Red Hat. VMware and Microsoft Hyper-V are proprietary, albeit free to use under certain circumstances and with some restrictions. KVM is the most recent hypervisor to emerge and has grown immensely[9] in popularity the last years. It is now considered the main hypervisor in all major Linux distributions, and is included in the mainline Linux kernel as the only hypervisor. This means easy installation and few dependencies, but it also means sustained development and quality assurances by

the Linux community. Since it also is the platform of choice among most distributions KVM is well documented, explained and has great compatibility with other applications and systems. KVM reuses many components from the Linux kernel and therefore benefits from performance improvements in the general kernel. KVM is solely a hardware virtualized hypervisor. A virtualization supported x86 system like modern Intel and AMD systems are an absolute requirement for KVM to function. For non fully virtualized components like IO devices KVM support a partial paravirtual driver, *virtio*, to improve performance.

Xen was a paravirtual hypervisor only until recently. It now has an option of full virtualization if the proper hardware support is present. Xen has for long been the most popular open source hypervisor, and the only fully paravirtual alternative. The trend has now shifted towards KVM and Xen usage is therefore slowly deceasing. This is mainly because it never was included in the Linux kernel and as a result is a lot more cumbersome to install and configure. Citrix acquisition of Xen has also contributed to many converting to KVM.
VMware is the best established commercial virtualization product on the marked. They have a range of products ranging from hypervisor to monitoring and management. Their hypervisor, ESX/i, is free to install and use, but has very limited management capabilities without additional products. KVM and VMware uses mainly full hardware virtualization, but supports certain devises to be paravirtualized for better performance.
Hyper-V is a hypervisor from Microsoft, aimed to compete in both the desktop and server virtualization marked. They advertise that it uses a fancy hybrid solution but is very similar to ESX/i. Hyper-V uses mainly hardware virtualization with the addition of drivers to increase device performance.

There are also other hypervisors, like VirtualBox, but they are not major players in server virtualization. VirtualBox has been acquired by Oracle and has support for both software and hardware virtualization with optional paravirtualized drivers.

2.1.3 Nested Virtualization

Nested virtualization is when one VM runs inside another VM that runs on the physical hardware, and this way creating chains of VMs. Full hardware virtualization requires support in the processor to host guests. Therefore when a standard hypervisor provide a generic CPU to its guest it normally does not include this functionality, which means the guest can not act as a hypervisor. However work are underway to enable this, and in the case of AMD processors the latest driver supports "pass-through" of the virtualization module. The hypervisor can then provide this functionality to its guests and enabling VMs to use the hardware accelerated VM-support. For Intel-CPUs this is only in the development stage and not yet ready for production. It is included in mainline Linux 3.2, but only for testing purposes. Full production-ready Intel support is expected soon and in time for the Ivy Bridge processor family

launch.

To nest virtual machines we can increase the flexibility and for example migrate entire clusters of VMs. But although this now is possible theoretically, it raises some performance concerns.

If the CPU-module is not available for some reason the hypervisor will fall back to emulation through QEMU. This will significantly reduce performance.

Figure 2.1 shows the basic layers in nested virtualization. Layer 0 is the physical hardware and the software running directly on top of it. That means a type 1 or type 2 hypervisor including the host operating system. Layer 1 is the first virtual layer. This is the virtual machine or guest hosted on the L0 hypervisor. In normal virtualization this is the top layer and where the applications and services are executing. But with nested virtualization this L1 also has a hypervisor installed. This hypervisor will provide a similar environment as the L0 hypervisor and enable virtual machines to be hosted inside the L1. L2 are the nested VMs as they are running on top of another virtual machine. The L2 VM will behave exactly as the L1 VM as there are no way to tell that it is nested. In this thesis the focus lies on two levels of virtualization to create a two layered cloud. There is no technical limit to how many nested levels that can be created, but as each layer will introduce an overhead large amount of recursion has limited practical use. Administration of such systems also becomes increasingly complex with the number of layers.

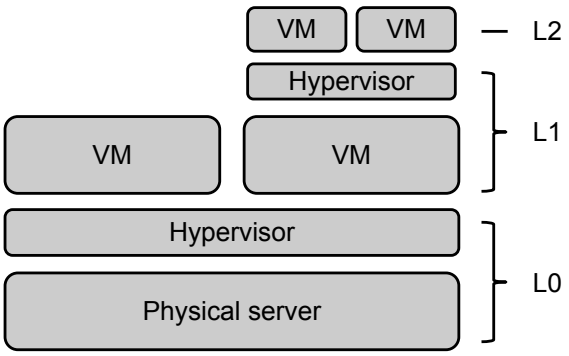

Figure 2.1: Nested virtualization

Nesting virtual machines, or recursive virtualization, was researched and described in a mainframe environment in the early seventies[22]. Nesting could be seen as useful in testing of virtual products under development and testing as this require extensive testing. Sandboxing applications inside virtual machines will also be possible at multiple levels. There has been some effort[5] into reducing the overhead of nested virtualization.

2.1.4 Live Migration

To migrate a virtual machine means to move it to another physical machine. When using *live migration* the movement is done without shutting the VM down or brake its connectivity. All the major hypervisors have support for live migration, but live migrating between different hypervisors is generally not possible.

When a VM is live migrated only the state and memory is copied over, not the entire disk. Therefore the two physical machines must also share the storage of the VMs hard drives, eg. NFS or iSCSI. The limiting factor for migration performance is the memory, so the more memory the VM uses and the more it changes during the migration the longer it will take. All memory pages will be copied over to the destination while the VM is still running on the source, and when this operation is finished it must re-copy all the dirty pages, the pages that has changed during the transfer, to the destination. Then the VM will be halted on the source and the remaining dirty memorypages are copied and the VM is resumed on the source. This will cause a short downtime, and is dependent on the activity during the transfer. Normally this downtime is just a few milliseconds and not noticeable by the user. If the downtime is short enough it is called seamless live migration, but it will never be 100% seamless since the dirty pages must be recopied at some point.

2.2 Cloud Computing

Cloud computing has been defined and described numerous times throughout history and the term is widely misused and customized to fit the product or service in question. The current definition on Wikipedia states: *Cloud computing is the delivery of computing as a service rather than a product, whereby shared resources, software, and information are provided to computers and other devices as a utility over a network*. This is an extremely wide and open definition and covers nearly the entire modern IT ecosystem. "The cloud" is also often divided into three categories based on the abstraction from the infrastructure and hardware.

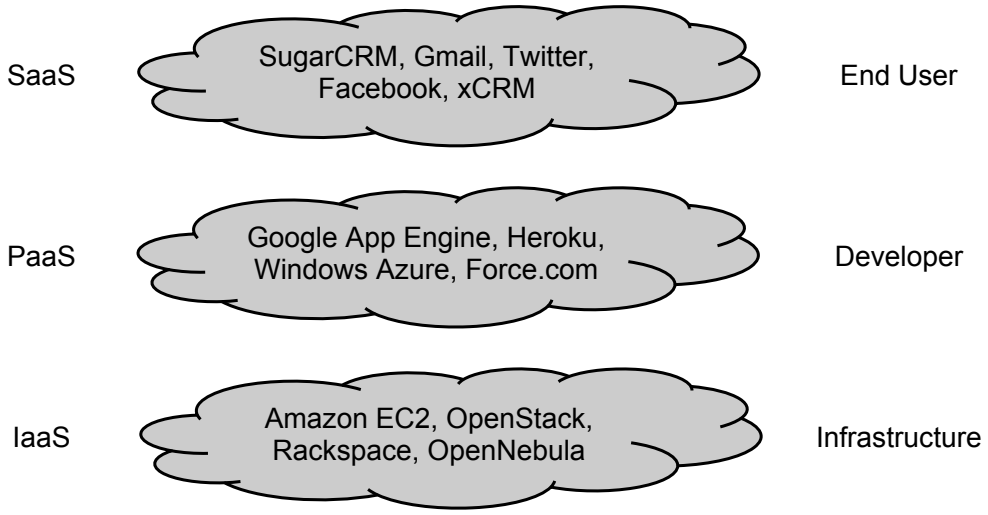

Figure 2.2: Cloud computing overview

- Software as a service. SaaS
  This is the most adopted type of cloud. Some argue that if this is considered cloud all websites and the entire internet is cloud computing. Software as a service is what we normally would call a *web-application*. Facebook, Gmail and Twitter are all considered SaaS. This is normal services the end user would use on a daily bases, but could also be services like content management systems (CMS) in use by companies, eg. a publishing solution for a web-based newspaper. The software are complete and often running off-site with a pay-as-you-use model. They also come with APIs to extend the functionality and customize to fit needs. This development is done as plugins or extensions to the application. This layer of cloud controls the content of a system. All underlying infrastructure and applications are in control of the provider.

- Platform as a Service. PaaS
  This is a development platform providing an API to host applications. Instead of offering an application to the customers it provides a platform to build applications on top. All infrastructure is owned and controlled by the providers and only small changes in configuration are allowed to the developers. The platform includes libraries and APIs to build the application. Google App Engine is an example of such service. PaaS is about controlling the application and logic of the system. The infrastructure and network is out of control of the customer.

- Infrastructure as a Service. IaaS
  IaaS has only one abstraction layer in contrast to the above services. The only layer the customers do not have control over is the physical hardware like servers, network hardware and hard disks. For many this is the "true" cloud computing. It is about hosting virtual machines off-site and on demand. Amazone was the first public providers of such a cloud ser-

vice on a large scale. They provided the physical hardware and an API for staring, stopping and basic manipulation of virtual machines. Beyond this the customers are in complete control of the operating system of choice and all the above layers. The configurations available to the users are mostly of a organizational nature like payment and basic allocation of virtual resources like the amount of memory, processor cores and storage space.

As this thesis will concentrate on the IaaS layer of cloud computing we will use the term cloud as the infrastructure as a service layer.

The cloud, that is IaaS, is a platform for running virtual machines. That is the abstract view, but the cloud platform itself is no more than a management layer above normal virtual machines running on a standard hypervisor. What the cloud does is to centrally manage multiple servers with hypervisors. We can consider the academic MLN (Manage Large Networks) project as a tool to manage groups of virtual machines. But it also supports running daemons on multiple servers to manage a network of virtual machines spanning numerous servers. It will therefore be possible to start, stop and live migrate VMs between a range of servers. By comparing MLN with a cloud platform there is no fundamental technical differences, so cloud computing is just a large scale environment for central management of virtual machines.

Beyond this basic operation of a cloud platform there are numerous different implementations and services provided on top for added functionality and optimization.

2.3 Amazon Elastic Compute Cloud

Amazon was the first large scale offering of public cloud computing on the IaaS layer. It was initially intended as an optimization of the management of their internal data centers. But they soon realized that their technology had potential as a public service and a profitable business model. In 2008 they launched the Amazon EC2 (Elastic Compute Cloud) as part of their AWS (Amazon Web Services). EC2 allows customers to launch *Amazon Machine Images* to create virtual machines running in Amazon-owned data centers. Amazon call these virtual machines *instances*. Amazon provides a range of standard AMIs with familiar operating systems such as Ubuntu, Red Hat, CentOS, Windows etc. But they also list community created AMIs where users have bundled an operating system with some software and configuration in ready to start servers. This can be complete LAMP web servers installations or commercial products installed by the providers for testing. When choosing an operating system to start they also provide some hardware choices. The amount of memory, the number of virtual processor units and the type of storage. There are two types of storage available. The first has no persistency, which means all data stored or changes made will be lost on rebooting. The

second method is called EBS or Elastic Block Store. This will give the user a certain amount of storage and it will act as a normal hard disk, resulting in persistency during reboot. EBS is a little more expensive. Recently Amazon also introduced a choice of virtualization type. Traditionally they only provided paravirtualization, but now also supports hardware virtualization. EC2 implements a slightly modified version of the Xen hypervisor.

The cost of running a single small instance in amazon EC2 is relatively small with a basic concept of paying for each hour the instance is running. But the complete billing formula is complex and includes the size of the instance (processor and memory), the storage type and size, the network traffic and many more factors from additional services like elastic IPs and monitoring.

EC2 now also has a large variety of extra services to optimize the experience. There are basic additions such as allocating a public IP to an instance to make it reachable from outside the cloud and various levels of monitoring and alerts to notify users of any abnormal events. Amazon is constantly adding and expanding its products and can now offer more complex functions like automatic load balancing, mail systems, database solutions, caching, queuing and object storage. All the products are running on the Amazon cloud and are specific for AWS users.

2.4 Public Cloud Security and Privacy issues

'A system is never more secure than its weakest link'. A cliché, but very suitable for security in cloud computing. And with increasing complexity and abstraction it is becoming a challenge to identify a 'week link', especially since cloud providers often hide layers and technology from the user for convenience and ease of use. The legal aspect is also increasingly outdated and ensures further difficulties. Legal concerns can often be considered the biggest security weakness. Traditionally computer security is ensured with three basic concepts [29], C.I.A:

- Confidentiality
- Integrity
- Availability

But in the age of cloud computing it is necessary to introduce at least two additional terms:

- Control
- Audit

Each of these categories have its own weaknesses, relevance and importance in the modern usage of cloud computer systems.

2.4.1 Availability

Availability is often a significant part of the motivation for a migration to the cloud. The cloud providers have large data centers and highly redundant architectures that provides good availability, and its very nature of delivering the services over the internet makes it reachable from anywhere. Most providers have some form of backup and protection against data loss, but this is also considered among the most significant worries [2]. The backup provided is in most cases not managed by the user and therefore harder to verify and control. This automation is build on trust of the provider and guarantees are not given. Self-healing and self-optimizing is tightly build into the cloud technology and on a large scale. This is a security risk in the way that data might be moved or reorganized for the benefit of performance or cost-savings by the provider. Equally the length and frequency backups are taken is controlled by the provider.

2.4.2 Confidentiality

Outsourcing a computing task or storage will always raise confidentiality questions, and even more so in a cloud context as the relationship between user and provider is more complex than traditional outsourcing. Often there will also be a third party, or service provider, between the cloud provider (the owner of the data center) and the user. The cloud system will have protection from most external attacks in the form of firewalls and intrusion detection systems, but again these are out of reach for the user and the systems must be trusted to keep attackers out and data in. Encryption is one possible solution for highly confidential data. When using the cloud solely for storage and backup the data can be encrypted before uploaded to the cloud, thereby hiding the content. This could be applied to medical records, financial information etc. The problem here is that such data can not be processed by the cloud, and will require a different computing service.

2.4.3 Integrity

As described in the section "Availability", cloud systems uses self-healing and optimizing techniques to cope with the growth in data and to maintain performance. By moving data around there is always a risk of corruption and loss. Normally this is solved with redundancy, backups and checksums. And again intrusion from both external and internal sources poses a threat to the integrity.

2.4.4 Control

Lack of control is the fundamental cause of most security and privacy issues in the cloud. The users of a cloud service has little or no control over physical hardware and software. This poses the question of trusting the provider. Even if the service fulfills all other security and privacy guidelines the lack of control

will remain a great concern. Data could be held hostage during a criminal investigation, data could be lost when the provider goes bankrupt etc.

2.4.5 Audit

Due to the control issues in a cloud computer environment auditing is essential to monitor and react to events that could threaten the integrity, confidentiality or privacy. A way to audit transactions at all levels and areas is still missing in the cloud architecture.

2.4.6 Geographical problems

Availability and scalability are common justifications for using a cloud based system, and to achieve these properties many public cloud providers use multiple data centers across different geographical locations, often continents. This raises numerous issues for users that depend upon confidentiality and control over the data. Very simplified cloud computing can be compared to outsourcing data storage, but while traditional outsourcing is fixed (both geographical and partners) cloud systems is by nature dynamic and complex. Data could be spread over multiple locations and involving countries and areas of different legal regimes. With todays cloud solutions there are no definitive way to control the data flow. Asian law can differ from European and American laws. The US has a liberal legal standpoint regarding surveillance compared to Europe which offers greater protection, but others might even enforce surveillance of data and/or traffic, eg. China. Besides surveillance there are a wide variety of potential problems; throttling and prioritizing of bandwidth [10], political change or instability, physical security and commercial prioritizing. All this can result in lack of control for end users of the cloud service. These issues are hugely complex and is yet to be fully solved.

2.4.7 Legal aspect

Privacy, licensing and copyright laws and regulations are well documented for business relationships when traditionally outsourcing storage and computing resources. Cloud computing however introduces a new type of relationship, three-party [39]. First we have the cloud provider, the company that owns and runs the data centers and host the cloud service. Secondly there is a service provider, a company that uses the cloud as a platform for their products and services. They are a customer to the cloud provider. And finally we have the users of the services and the cloud. Because cloud services often uses this three-party model laws and regulations are not necessarily applicable and the end users have very little legal rights. To further complicate the issue the cloud provider operates in different locations [27] and countries with different laws as discussed above.

2.5 Private clouds

A *private cloud* is a set of software used to manage an infrastructure where the user are in control and owns the hardware. Modern private cloud solutions have become so complete that they offers almost all functionality of the public clouds. They can even have similar user interfaces and systems for customers to pay for the use. This means that a private cloud can be used to setup a public cloud offering to others.

Despite the power a private cloud can have and the advanced functions it introduces it is conceptually a simple organization tool for virtual machines. All clouds need a hypervisor platform at the bottom of every node and the cloud software keeps track of the virtual machines running on top of those nodes. The possibilities offered by the hypervisor is implemented in the cloud API and is therefore centrally managed.

Private cloud could be used to organize existing virtual environments. But they could also be an alternative for public clouds because of the potential increase in security and privacy. A private cloud is owned by the user, and this means full control over all aspects and layers of the cloud, including the physical infrastructure. This simplifies the legal difficulties and the data ownership questions. Furthermore it gives freedom to design the right backup and redundancy solutions. Even though security could be better in a public cloud a private cloud will give full transparency and be implemented with the companies quality control.

The choice between using a public or a private cloud has many variables, and depends on what the system will be used for. There are two typical scenarios. One is to use the cloud to host a service available to the public. This could be a web service where normal end-users log in or access the content. The other is where the system is used for internal purposes. The services and content hosted in such a cloud is not accessible to the normal user outside the company. There are other cases as well, eg. making a public cloud for customers to host virtual machines on, much like Amazon EC2. But the two scenarios that most often will be adopted are for a public service or internal system. These two have different requirements.

One disadvantage of public clouds are their locations. If the service intended to run on the cloud has to be fast and responsive the public cloud might not be adequate as it could be to distant and with to high latency. This mostly applies when the cloud is for internal use. Internal use of a cloud will often involve confidential data and data not intended for the public. This might be reason enough to prefer an internal private cloud.

On the other hand a public service like a web service could be considered better suited in a public cloud like Amazon EC2. Whatever the reasoning, the issues comes down to trust in the public cloud provider.

2.5.1 Private Cloud Products

There is a growing number of solutions and products to implement a private cloud. This thesis will mainly focus and use OpenNebula and OpenStack. OpenNebula was started as a research project in 2005, but was made publicly available as open source in 2008. OpenNebula is not to be confused with Nebula, which is a cloud platform developed by NASA and later became part of the OpenStack project. OpenNebula is community driven and under the Apache licence with CERN as one of the high profile users. The product is a framework, or toolkit, to build and manage private, public or hybrid clouds. The cloud consists of two main components, the front-end and the nodes. The nodes runs the hypervisor and hosts the virtual machines and the front-end manages the cloud and serves the API. OpenNebula supports multiple hypervisors, KVM, Xen and VMware.

OpenStack provides the same basic service as OpenNebula. It is a cloud framework to build private IaaS clouds. OpenStack was initially a joint effort between NASA and Rackspace. NASA provided the technology for hosting and managing virtual machines, the Nebula project, and Rackspace provided the storage system in the form of Rackspace Cloud Files. The OpenStack project has now over 150 major business contributors including AMD, Intel, IBM, HP, Dell, Cisco, Red Had and Canonical.

OpenStack is build up of several components. Nova is the cloud computing fabric controller, which means the organization and hosting of virtual machines. Nova is the previously known NASA Nebula. Swift is an Object storage system for redundant cloud storage. Glance is the image service that manages the images used to launch virtual machines, this is the successor of Rackspace Cloud Files. In addition to these core services, OpenStack includes Keystone, a identity manager, and Horizon. Horizon is a graphical front-end running as a web server to manage the cloud.

Chapter 3

Model and Methodology

This chapter will explore the concepts used to achieve the goals stated in the introduction. The limitations in current systems will be highlighted and followed by ideas and theories to solve them. A theoretical system which will be explained and an implementation suggested. This system architecture will be the reference for the proof-of-concept implementation and testing platform used later in the thesis. The tests and benchmarks will also be set up and prepared to run on the system to best form conclusive results.

3.1 Concept

3.1.1 Problem

Cloud Computing is generally viewed as a flexible and elastic platform for modern IT-systems as they provide on-demand computing with on-demand cost. All large public cloud providers will advertise scalability, flexibility and openness as some of their top advantages. Since they give the customers total control over the VMs they claim no lock-in and that portability are in the hands of the developers, not the provider. However, cloud portability and a practical way to avoid becoming dependent upon a cloud technology is still a problem. Services on top of clouds are often tied into the API that hosts the VMs. Eg. a number of web servers being hosted in the cloud behind a load balancer to dynamically scale depending on the demand and incoming requests. To balance the load many providers have proprietary solutions like Amazon Elastic Load Balancing[1]. But even if the load balancer resides inside a VM controlled by the customer it needs to communicate with the cloud API to start new VMs or perform other operations. This simple example shows that even a small and simple service that does not depend on many advanced cloud features use cloud specific solutions and therefore tie their code to that provider. Even in private clouds, where the customer own and control all aspects of the cloud and infrastructure the API is still specific to the cloud technology in use.

The virtual machines running in the private cloud can be migrated freely around

[1] http://aws.amazon.com/elasticloadbalancing/

inside that cloud, and this is supported by all major private cloud systems. But the VMs can not leave the cloud environment and be migrated to a different platform. A VM running in OpenNebula could not be migrated to an OpenStack cloud, even if they where on the same network. The same applies to public clouds, where an Amazon EC2 instance not can be migrated to a RackSpace cloud. There are hybrid clouds which can combine two or more cloud platforms, but all they do are managing multiple clouds with a single interface. A hybrid cloud does not enable migration of VMs between the clouds. These limitation means that the customer mush choose a platform, and that any change in that choice will result in a major converting of platforms.

So when cloud computing is thought of as flexible and elastic it means elasticity for the customer and scaling the number of VMs up and down. This is true for both public clouds like Amazon and private clouds like OpenNabula. But the cloud platform itself is not very flexible. Since it is installed on physical hardware it cannot be migrated or scaled easily without physically moving server racks or acquire more hardware. This is partly the reason why inter-cloud migration is difficult in todays systems. The VMs running in the clouds are hosted on the hypervisor installed on the physical server. These hypervisors are not compatible with each other, and will therefore hinder any migration. So if inter-cloud migration where to work in todays architectures all hypervisors must have been compatible and all cloud platforms must have shared information about the VMs.

3.1.2 Theory and Solution

The goal is more flexibility and to decouple the cloud from the physical hardware. A basic private cloud consist of two main components; administrative and compute. Administrative means the systems to control the behavior of the cloud and the virtual machines running on it. Typically a cloud controller or front-end storage. These could again be divided into multiple components depending on the scale of the system and performance requirements, front-end web gui, API-server, queuing-server, database, scheduler etc. The storage is often not considered as it could be a separate system, eg. a SAN. All of this is to control the cloud and could in most cases be hosted on the same server. The other big component is the cloud nodes. They can also be called compute nodes or workers and are servers with hypervisors and with generally high amount of processors and memory to host the virtual machines. This is mainly where the potential lies. A compute node has a hypervisor installed and will start VMs on request from the controller. The guest VM can then be migrated to another compute node, but the node itself is installed directly on the nodes physical disk and therefore static.

By introducing a virtual layer between the physical node and the hypervisor running the guest VMs the compute node is decoupled from the server.

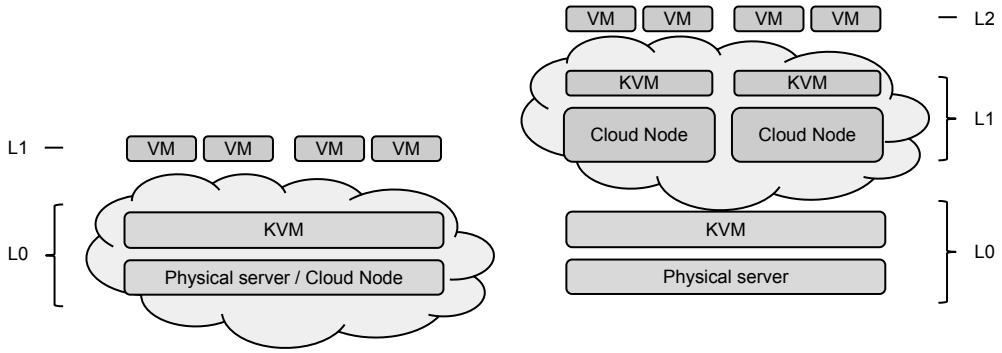

Figure 3.1: Traditional cloud and new virtual cloud

Figure 3.1 illustrates the introduction of an additional layer. The left side is a traditional cloud implementation with the hypervisor installed directly on the server as L0. The guest VMs will then run on top in layer L1.

On the right side the cloud node is not installed on the physical server but runs inside a normal virtual machine. The cloud node has therefore become the L1 layer VM. To host any further virtual machines we now need a second hypervisor, a nested hypervisor on layer L1. The installation of this virtual cloud node is identical to the traditional L0 node. The cloud will therefore behave the same way. The guest VMs will now be hosted on top of the second, nested, hypervisor on layer L2. The VMs also operates as before, with a potential overhead as the only difference. We can now say that the cloud itself has become virtualized.

Since the cloud nodes and cloud guest VMs function the same as they did before, normal migration is still possible. Figure 3.2 shows a normal live migration of a guest VM from one node to another inside the same cloud.

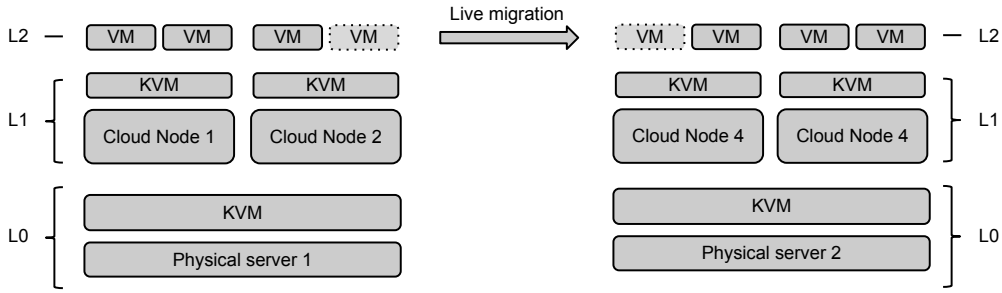

Figure 3.2: Live migration of guest VM

Since the nested hypervisors and the virtual cloud nodes are now decoupled from the physical servers, they will inherit all properties and flexible attributes of a normal virtual machine. This includes the ability to be migrated.

When a cloud node is migrated from one physical server to another it will retain its guest VMs, figure 3.3. All guest VMs hosted on that cloud node are therefore also migrate to the new physical server. It will also work on live migration since the memory of the nested guest VMs are included in the cloud nodes memory that are migrated.

This will also open the possibilities to easily migrate cloud nodes with guest VMs to different types of hardware. One can imagine moving a node to an idle workstation to utilize the resources, or in scenarios where there are few guest VMs running they can be migrated to more power efficient servers or workstation if that is sufficient to maintain operation.

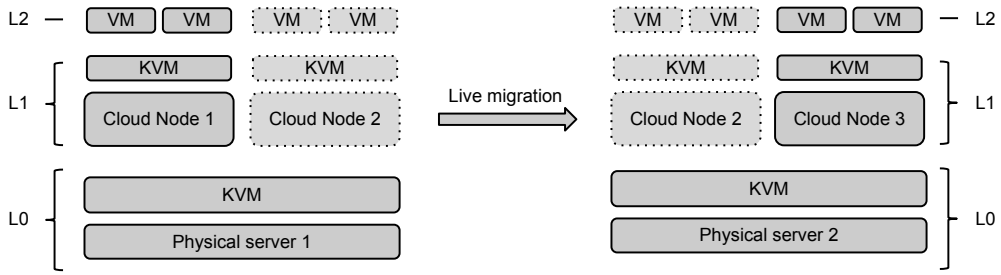

Figure 3.3: Live migration of whole cloud node, including its guests

The ability to live migrate hypervisors including the internal virtual machines opens up a wealth of opportunities. The first, and most obvious, is the advantages of not being tied to the physical servers. But one can also imagine a scenario where the new, virtual, cloud can be live migrated over to a totally new physical infrastructure.

In public clouds the user will normally not have control over the hypervisor. This is a limitation to what can be done with the guest in the sense of configuration and migration. An essential requirement for live migration is a shared storage. This is clearly not possible in a public cloud since the hypervisor is out of our control and therefore also the storage of our VMs. The VMs are stored where the provider has configured, not where we want for live migration to work.
In a setup with nested virtualization the administrator has control of the L1 hypervisor, which gives us the opportunity to choose the storage method of the nested, L2, virtual machines.

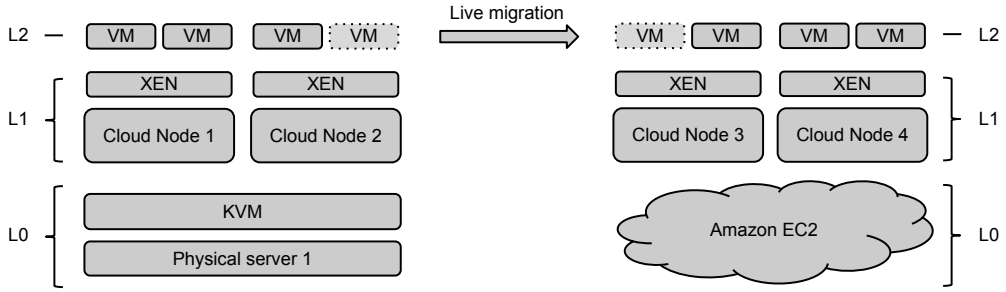

Figure 3.4: Live migrating guest VMs to a public cloud

When control over the hypervisor at layer L1 is achieved the user can create his own storage solution that fulfils the shared storage requirement for live migration. This storage could be in the public cloud or in a separate external system controlled by the administrator. Multiple nested hypervisors can mount this same shared storage and create an environment that enables live migration of the L2 nesten virtual machines between public cloud instances. This can also be expanded to different public clouds or combined with private clouds that shares the same storage. Figure 3.4 shows a private cloud live migrating its nested VMs to a cloud node running in an Amazon EC2 instance. The cloud nodes hosted in the private cloud and the cloud nodes running in the public cloud are both members of the same private cloud spanning over the two underlying services. With a more abstract view, figure 3.5, shows that this is one private cloud running inside and spanning over multiple private or public clouds with the ability to live migrate the guest virtual machines between all the services. The new abstraction from the L0 clouds also eliminates the hypervisor technology they use. As long as fully nested virtualization is supported the L0 hypervisors will not affect the migration of L2 VMs between them.

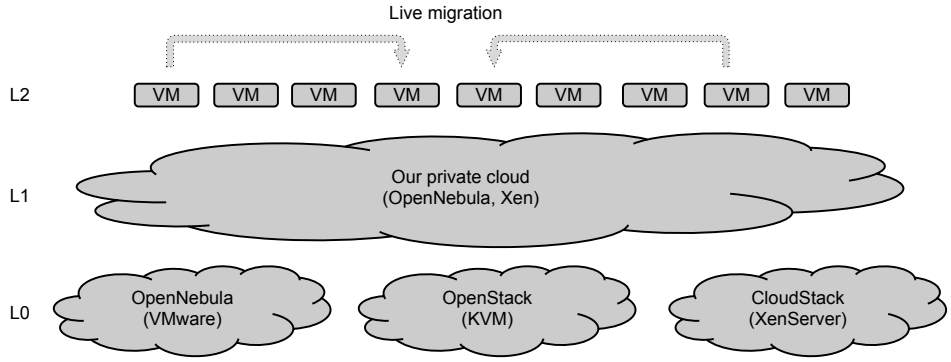

Figure 3.5: Virtual cloud spanning multiple heterogeneous clouds

3.2 Platform and environment

Here follows a description of the environment used throughout this paper and how it will effect the experiments.

3.2.1 Base Environment

For the tests and implementations a range of hardware will be used to simulate the most significant scenarios and setups. The first set of servers are Dell PowerEdge servers each equipped with two Quad-Core AMD Opteron 2.8GHz processor, 32GB memory, two 1Gbit/s network cards and a Seagate 500GB 7200rpm hard disk. In addition a number of standard HP desktop computers, acting both as servers and as workstations, will be used. They are fitted with a Dual-Core Intel Core 2 at 2.13GHz, 4GB memory, a 1Gbit/s network card and a Hitachi 160GB hard disk. All servers are connected through several gigabit switches. Figure 3.6 shows an overview over the environment.

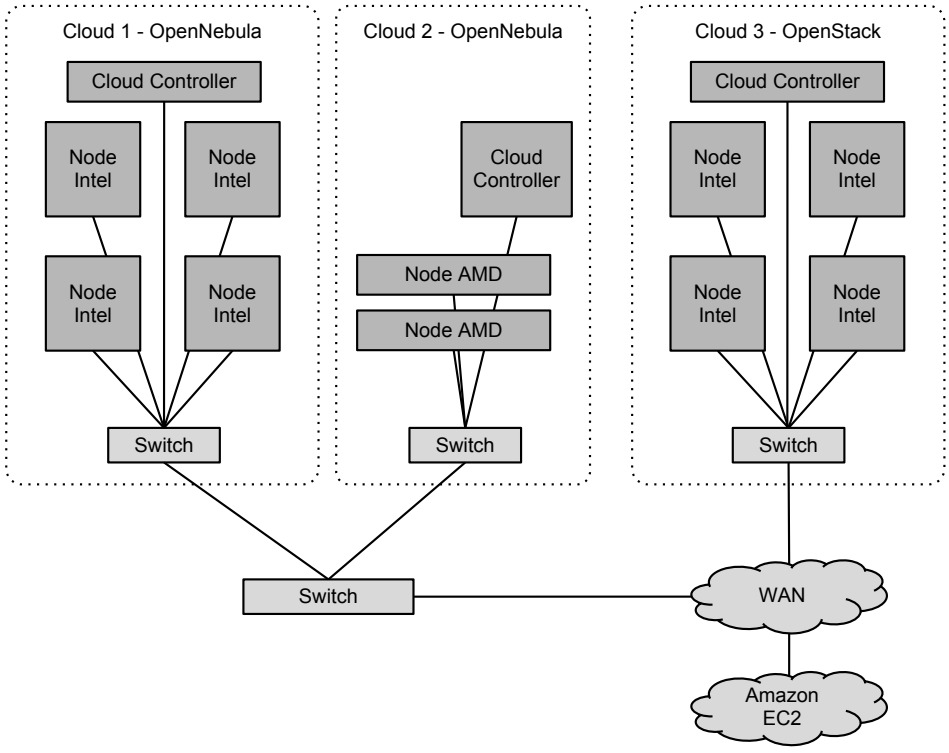

Figure 3.6: Physical setup

All physical servers and workstations will use the 64bit Ubuntu operating system. For the hypervisors this will be a minimal Ubuntu installation. Since the paper is investigating if the proposed architecture is a feasible method of hosting virtual machines be eg. measuring the overhead limit the resources used by the operating system itself is essential. The only difference in setup will be the relevant software like hypervisors and cloud platforms, all other

components will be the same to eliminate variations and potential void the results.

3.2.2 Hypervisor

In this paper the focus will mainly be on the KVM (Kernel Virtual Machine) hypervisor. KVM is integrated with the modern mainline Linux kernel and requires minimal configuration. As support for nested virtualization on both AMD and Intel processors are needed, a combination of Ubuntu 11.10 and Ubuntu 12.04 Beta will be used. This is mainly because of the kernel version, Linux 3.0 on Ubuntu 11.10 and Linux 3.2 on Ubuntu 12.04. Linux 3.2 could also be installed on previous Ubuntu-versions to enable nested VMx support or it could be compiled into the kernel for all mainline versions after 2.6.38. To enable nested virtualization an argument must be passed to the virtualization module when loading it into the kernel, respectively:

```
kvm_amd nested=1
kvm_intel nested=1
```

For nested virtualization to function it requires a 64bit kernel, so all systems will use 64bit operating systems.

3.2.3 Storage - NFS

For live migration to work all nodes must share image storage. When a VM is migrated only the memory is copied, the disk must be mounted from the same source for both source and destination to avoid huge amount of data being transfered. But since introducing a new virtualization layer the storage becomes a little more complicated. Both hypervisor layers need a shared storage. The first layer will store all images of the cloud node VMs on the first layer cloud controller. This will enable us to live migrate the cloud nodes to any location with access to the storage. The second layer of storage is for the cloud guests running inside the cloud node VMs. This storage will be managed by the cloud controller, eg. OpenNebula, and enables the guest VMs to be live migrated between the cloud nodes. The actual storage are two NFS shares hosted on an external system, this could be a SAN or any other storage system. An alternative is to use a virtual cloud node to host the shared storage. This will enable us to also migrate the storage server, but it might require massive data transfers and therefore not be a practical solution.

3.2.4 OpenNebula

For the setup in this paper the OpenNebula installation will be divided in two parts, the front-end and the compute nodes. The front-end will also host the database, API, webinterface and all other components of OpenNebula, except the compute nodes. This is done for simplicity and will not effect the performance and scalability in the experiments since this is on a relative small scale,

up to ten physical nodes.

The compute nodes has a very simple setup in OpenNebula, all they essentially need are a hypervisor and an SSH server. But for the system to function properly a few more parts are needed. To mount the shared NFS storage they need the nfs-common package and an entry in /etc/fstab to mount at boot-time. It is also important that all nodes and front-ends have synchronized clocks, so a NTP client is required. For live migration to work all nodes must be able to resolve all other nodes so a working DNS system is important. Equally they must be able to access each other through SSH without password so all keys must be exchanged. Finally the hypervisors and its managements tools like libvirt and QEMU needs a few tweaks to let OpenNebula access and modify them.
The full installation documentation is available at opennebula.org[2].

3.3 Measurements

3.3.1 Motivation

For this theory of an added layer of virtualization to be feasible it is essential that the additional overhead is at a minimum. If the system experiences a significant performance drop when virtualized, the added flexibility will not be justify as more resources are needed to maintain performance. All virtual systems have some kind of performance loss compared to the native system. The emulated environment presented to the virtual machine will never perform as well as the native components. But although all virtual systems have a performance overhead it is still widely used in the industry, so for most systems this is an acceptable sacrifice to gain functionality. When introducing a new virtualized layer inside the existing one there will naturally be a new layer of overhead with potential performance loss. The nature of this overhead is determined by how the two hypervisors interact with each other. The virtualization extensions implemented by Intel (VT-x) and AMD (AMD-V) works in a similar way, although the implementation is different[7], by only allowing one hypervisor to use the extension. This means that the first layer hypervisor will handle all traps occurring in all the layers above. This could potentially decrease the overhead each virtualized hypervisor represents.

There are three main limiting performance factors of interest:

- Computation performance
  This will dictate how fast the virtual machines can operate. This can for example be a mathematical calculation or processing a web request.
- Disk IO performance
  The disk performance determines the speed at which files can be ac-

[2]http://opennebula.org/documentation:rel3.4

cessed. Both read and write of large sequential file and random seek time is essential for overall performance. A potential problem area regarding disk performance is the network. Because the storage is central all disk operations travels over the network, and can be affected by the network performance.

- Network throughput and latency
  Network performance is essential for most services and also for migration and administration of the cloud. Both throughput and latency can affect the overall reactiveness of the system.

3.3.2 Collection

All hypervisors and guests will be kept at standard out-of-the-box configurations to be best comparable. The tests will be performed on a cloud guest VM running traditionally on a single hypervisor and then compared to the same cloud guest now running inside a second, nested, hypervisor on the same physical server. The performance delta will indicate the overhead introduced by nesting the virtual machines.

The benchmarks will be synthetic tests to get the most accurate and isolated measurements. The following tools and methods will be used:

- Computation performance
  This test will measure the raw processing power of the system, in number of calculations per second. The test itself will be a script calculating the mathematical constant pi. The formula used is Leibniz[3][6] formula for pi.

$$\frac{\pi}{4} = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} \tag{3.1}$$

  This algorithm is known for being precise after many iterations, but converges slowly and is therefore generating high load on the processor over a long time. It will also give very consistent results when executed multiple times, so it is ideal for benchmarking the CPU with high load. The formula will iterate 10,000,000 times which gives a pi accurate to 6 decimals. The calculations will start over and repeat 200 times to become statistically significant and representative for the CPUs performance. Then an average over the 200 samples will be recorded. The script is attached in the appendix.

- Disk IO
  The Bonnie test suite will be used for disk testing. To measure hard disk performance three factors are considered. Maximum write performance, maximum read performance and random seek. Combined these three figures provides a good indication of how disk operations performs.

[3] http://en.wikipedia.org/wiki/Leibniz_formula_for_pi

The most representative figures are the sequential reading and writing of blocks. The tests will be executed overnight without an internet connection or any other interference or disk usage.

- Network IO
  The network performance is divided into throughput and latency. Throughput is the maximum amount of data transfered over the network per second. The latency is the responsiveness of the system, how many packets that can be replied over a time. The network hardware are all 1Gbit/s network interfaces and for these tests the servers are connected directly to each other to eliminate any restrictions or delay in the switch. The tests will be performed with the *netperf* tool for both throughput and latency. The system is totally isolated and the tests are executed overnight.

3.3.3 Target performance

The whole concept and idea is motivated by increased flexibility and portability. And when introducing additional abstraction layers it is natural that an added overhead is seen. A performance drop is therefore expected compared to the standard cloud setup. If the loss is significant the added flexibility will not be justifiable. Some extensive benchmarks[21] of KVM virtualization on Intel and AMD shows generally 90% of native performance. We will be looking at a performance cut-off at around 10%. That means if the performance is kept inside 10% of the original performance it could be considered a non-significant sacrifice for the added flexibility. This cut-off will vary depending on the system requirements and the type of applications hosted on the platform.

Chapter 4

Results

In this chapter the results of the implementation of the concept is presented. The first part is about how the technologies and systems behaved and to what extent the theory will be practical and feasible. Later on the test results are presented and explained and lastly we explore the flexibility of our setup.

4.1 Nested virtualization

The installation of a virtualization platform utilizing nested virtualization is not as streamlined as desired. The hypervisor itself is well tested and proven and therefore easily installed, but the more recent drivers for nested virtualization are either not activated as default in the kernel or not included at all. In the case of AMDs module the nested driver is included but not turned on, but for Intel the adoption into the mainline kernel is as recent as 2012 and are not yet shipping in any major distributions. However the feature is present in the upstream Linux 3.2 and onwards, but for testing purposes only and not activated by default. This functionality could potentially be backported to earlier kernels if necessary.

When the driver is present in the kernel it is activated by passing nested=1 as an argument when loading the respective module (kvm_intel or kvm_amd) into the kernel.
To verify the presens of the nesting capability and that it is activated we can issue the following command:

```
modinfo kvm_intel | grep nested
cat /sys/module/kvm_intel/parameters/nested
```

```
modinfo kvm_amd | grep nested
cat /sys/module/kvm_amd/parameters/nested
```

These commands should return *nested:bool* on the Intel architecture and *nested:int* for AMD. The file should contain either Y or 1 when nesting is turned on and

functional in the current kernel.

When we want to check if a machine has access to hardware accelerated virtualization in the CPU we can issue the *kvm-ok* -command or verify that /dev/kvm exists. On a normal physical server that supports virtualization this is common, but inside a virtual machine this will return false since the guest VM does not have direct access to the CPU, but only sees the emulated hardware presented by the hypervisor. When the hypervisor activates the nested driver it can present the VMs with an emulated CPU that does support hardware virtualization. This can be seen when executing the command "lshw -c cpu". This will print out information about the CPU. Most significantly we notice that it is a *QEMU Virtual CPU* and that under *capabilities* we find the vmx or svm support indicating that hardware virtualization is present. This mvx or svm support in the virtual machine is created by QEMU when emulating the CPU.

In some versions of the nested driver or libvirt/qemu this vmx or svm capabilities is not inserted into new virtual machines on creation by default. We must then add these either with flags on VM launch or by editing the VM after creation and rebooting it. The configuration is found in the appendix.

When the nested driver is correctly installed into the kernel, activated in the hypervisor and QEMU is setup to emulate CPUs with a vmx or svm capable CPU, we are able to create virtual machines with access to hardware accelerated virtualization. Inside that VM we can check this be executing "lshw -c cpu":

```
  *-cpu:0
       product: QEMU Virtual CPU version 1.0
       vendor: Intel Corp.
       physical id: 1
       bus info: cpu@0
       width: 64 bits
       capabilities: fpu fpu_exception wp de pse tsc msr pae mce
cx8 apic sep mtrr pge mca cmov pse36 clflush mmx fxsr sse sse2
syscall nx x86-64 rep_good nopl pni vmx cx16 hypervisor lahf_lm
  *-cpu:1
       product: QEMU Virtual CPU version 1.0
       vendor: Intel Corp.
       physical id: 2
       bus info: cpu@1
       width: 64 bits
       capabilities: fpu fpu_exception wp de pse tsc msr pae mce
cx8 apic sep mtrr pge mca cmov pse36 clflush mmx fxsr sse sse2
syscall nx x86-64 rep_good nopl pni vmx cx16 hypervisor lahf_lm
```

Here we can see the two QEMU emulated Intel CPUs with vmx support. This is a clear indication that the virtual machine can host new virtual machines inside it with full hardware acceleration. We can further check that the hypervisor recognizes the support with kvm-ok:

```
INFO: /dev/kvm exists
KVM acceleration can be used
```

Virtual machines can now be started inside the nested hypervisor.

4.2 Synthetic Benchmarks

The tests were executed in total isolation with no running processes besides the ones critical for the tests and no outside disturbance. To obtain statistically significant results the tests were repeated at least 30 times. 30 samples is considered sufficient for the central limit theorem[1] to take effect. All tests are executed in an OpenNebula private cloud environment, and for the nested test both layers of cloud are OpenNebula.

4.2.1 Processor

We start with a processor test. This will show us the raw processing power of the virtual machines. The tests where executed on all three virtualization layers, whereas native is on layer L0 without any VMs. "First VM" shows the performance of a single, L1, traditional virtual machine, and the "Second VM" is the nested L2 VM. The Y-axis are for how long it took to calculate pi. The calculations where repeated 200 times and a mean where then calculated. The maximum and minimum values are also displayed as error bars on the diagram. The lower the value the better.

The first test is on the HP machine with an Intel dual-core processor as described in chapter 2.

[1] http://en.wikipedia.org/wiki/Central_limit_theorem

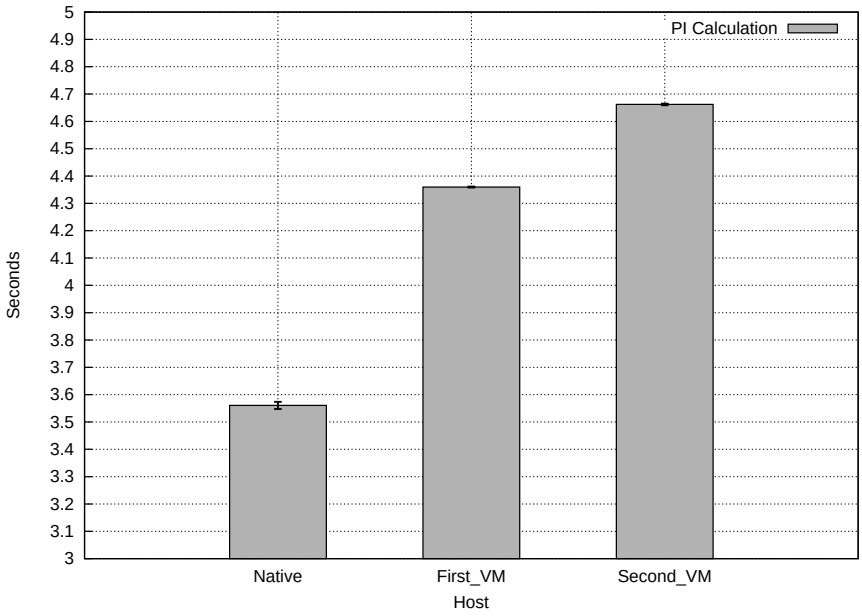

Figure 4.1: pi calculations on HP test-rig

Figure 4.1 shows the result of the first test. We can clearly see that when the test executes on the native machine without any virtualization it performs best. It is also very consistent as the min and max values are close to identical with the mean.

The results for the first virtual machine are significantly worse. The test took 4.35 seconds when running in the virtual machine compared to only 3.56 when executing natively. That is a performance overhead of 18.1 percent. But the second, nested, virtual machine running inside the first VM scores 4.66 which is only an additional 6.6 percent overhead. Compared to the native host the two layers of virtualization have added 18.1% and 23.6% respectively in performance overhead.

An additional observation is that the values becomes slightly more inconsistent with each layer of virtualization. And the best values from the second, L2, VM tests are very close to the worst values of the test in a single VM.
The same three tests where run on the AMD powered Dell system. First a baseline native test executing on the server without any virtualization. Then the two virtual machines, the second nested inside the first.

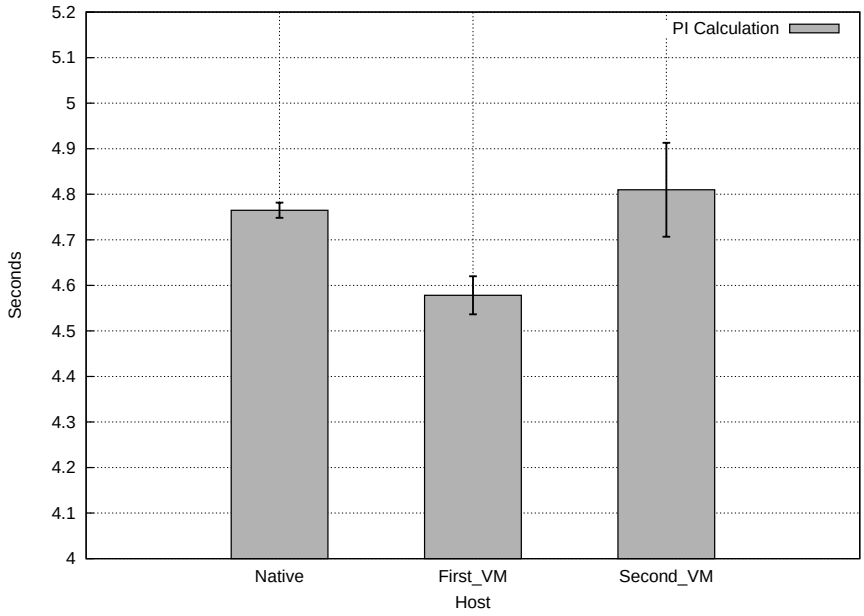

Figure 4.2: PI calculations on DELL test-rig

As we see in figure 4.2 the native server uses 4.84 seconds to calculate pi. It makes sense that the AMD processor is slightly slower than the newer Intel CPU, even though AMD has more CPU cores. This comes from that the pi script only utilizes one core in the calculations. The results from the virtual machines are however surprising. At 4.57 seconds and 4.81 seconds for the first VM and the nested VM respectively they performs better than on the native server. A virtual machine should in theory not be able to outperform its native host. To eliminate any sources of error in the system it was purged and reinstalled with a clean Ubuntu 11.10 64bit server edition fully up to date and without any additional packages except the SSH server. The native test was executed again with the exact same results. Then the KVM with QEMU and libvirt were installed and a simple VM was booted. The results were identical as before, and faster than the native server. The cause of this anomaly requires further investigation.

When only considering the two virtual machines, the normal and the nested, we can again see the small overhead as on the Intel powered HP server. The additional overhead when using a nested VM is only 4.7%. We can also see the same trend that the results become more inconsistent the more layers we add, but generally the results are good and conclusive.

4.2.2 Network

Now we will look at the network performance. How a network performs can be measured a number of different ways, and we will concentrate on two highly relevant values throughput and latency.

For all the network tests we will be using a 1 Gigabit/s switch between the servers, both servers with a 1 Gigabit/s network card. No other computers are connected to the network, and the server we are testing from does not use any virtual machines.

Throughput

Throughput is the rate at with data can be transfered from one host to another. In other words the maximum speed of a transfer. We measure the throughput in Mbit per seconds, Mbit/s.
The tests were executed with netperf and the command:

```
netperf -H HOST
```

The first test is with the HP test servers, and the three tests are the same as with the CPU-test. The first test was on the native server without virtualization. The first VM is a normal virtual machine and the second VM is the nested running inside the first.

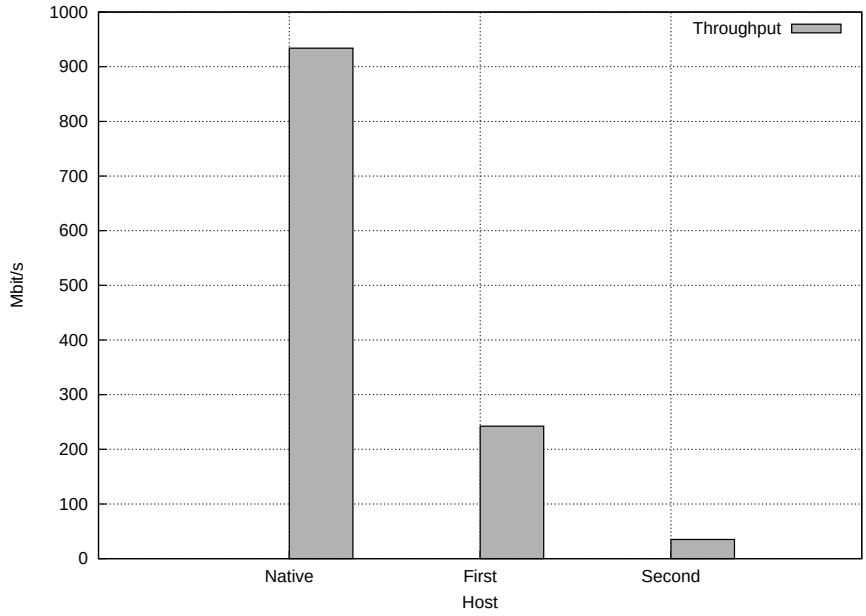

Figure 4.3: Network Throughput on HP test-rig

Figure 4.3 shows that the native servers performs as expected and very close to the theoretical 1 Gigabit at 934.04 Mbit/s. But for the first layer of virtualization there is a massive performance loss. At 242.46 Mbit/s it is a only 25.9% of the native performance. A similar dramatic overhead is observed with the nested second virtual machine. It clocked in on just 35.02 Mbit/s, which is totally unusable.

There could however be a good reason for this poor performance. The standard KVM setup for virtual machines is full hardware virtualization for the processor and the memory management, but the I/O device are emulated and passes though the extra layer with potential dramatic overhead. Xen solves this with paravirtualization of all drivers, including the network. And although KVM does not support full paravirtualization it supports a hybrid solution with special drivers for critical I/O devices such as network and disk. This will give KVM guests the advantages of paravirtualized performance rather than emulated. The specific driver is called *virtio*. In figure 4.4 illustrates the same test when using the virtio driver.

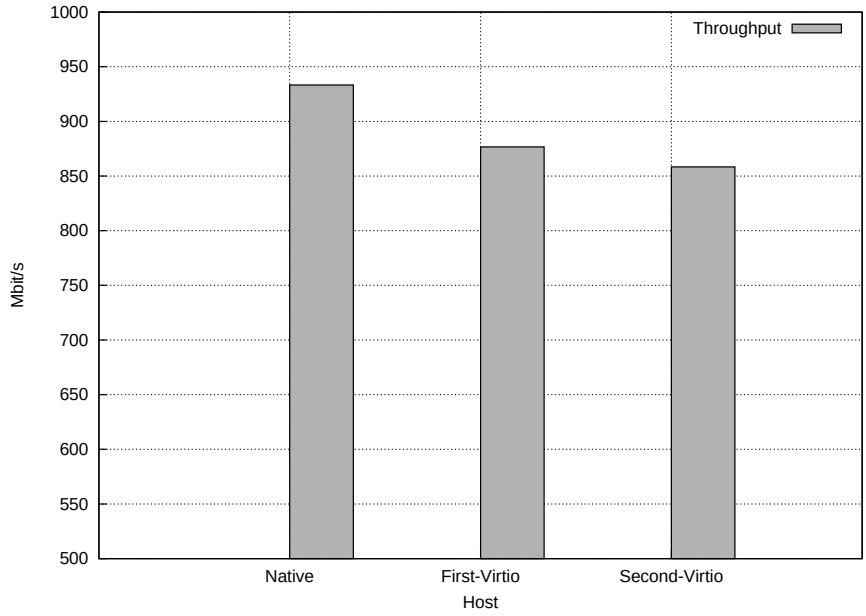

Figure 4.4: Network Throughput on HP test-rig with virtio

The native performance will remain at the same level as previously with 933.27 Mbit/s. But now a tremendous increase in virtual machine performance is observed. The first layer of virtualization now has a network throughput of 876.62 Mbit/s. This is only 6.1% slower than the native server. As for the second layer an even greater improvement is seen as the virtio driver now is active on both layers of virtual machines. The nested VM now has a result of 858.36 Mbit/s, which is a barely noticeable 2.0% slower than the first virtual layer. The end result is therefor a performance overhead of 6.1% with one layer of virtualization and 8.1% with two, one nested inside the first.

The next diagram, figure 4.5, are the results from the same network test as the previous, measuring throughput. But this time the tests were run on the AMD powered Dell server. As with the first HP network test this is without using the virtio paravirtualized driver.

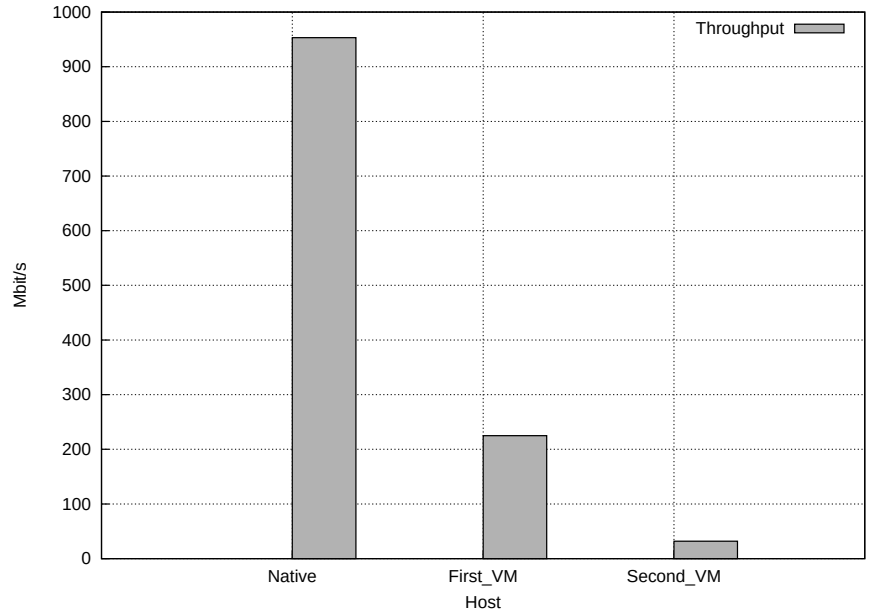

Figure 4.5: Network Throughput on DELL test-rig

As expected a similar trend is seen leading to a dramatic performance loss when virtualized. And the second layer of virtualization inherits the overhead of the first layer in addition to its own overhead, resulting in catastrophic performance. The values here are 953.14 Mbit/s on the native, non-virtualized, server. On the first layer virtual machine we get a measured throughput of 224.95 Mbit/s. The second layer have a maximum speed of only 31.83 Mbit/s. These results are close to identical to the comparable values on the other test-rig. The limiting factor is clearly the emulated network device.

When the *virtio* drivers are activated in the virtual machines the same significant improvement is achieved. Figure 4.6 visualizes the importance of this hybrid compromise of full and paravirtualization.

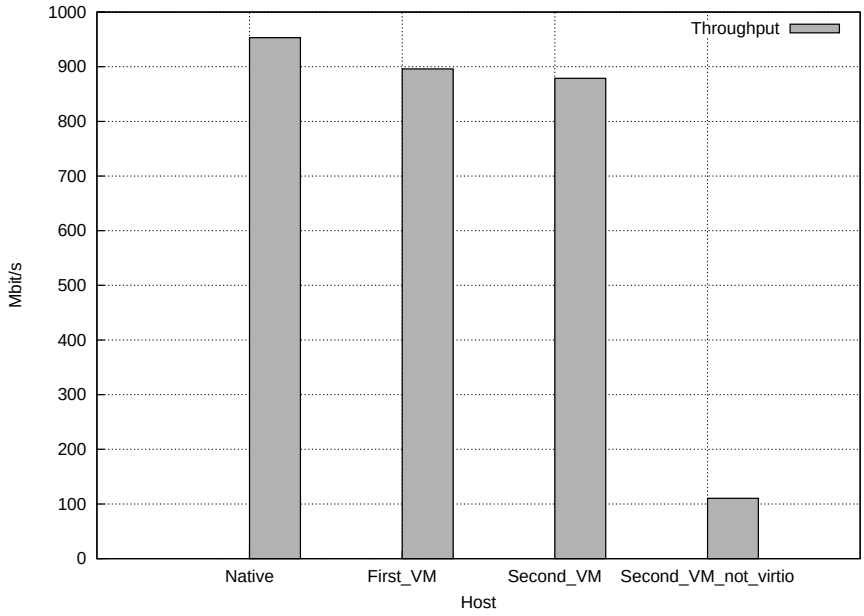

Figure 4.6: Network Throughput on DELL test-rig with virtio driver

The first bar shows the native performance of 955.07 Mbit/s. The first, L1, layer virtual machine is now at 896.10 Mbit/s, a loss of just 11.0%. And the second, L2, nested VM indicates 878.82 Mbit/s. That is an impressively small 1.9% additional overhead when nested. This is when both the first and second layer virtual machines make use of the paravirtualized driver. The fourth bar on this figure is added for comparison when the virtio driver is only used at the first level. The second VM will now use the default emulated network device and therefore suffers the performance hit.

Latency

The propose of a latency test is to measure the responsiveness of the network. Each component in a network will add a small delay to the traffic. Equally each layer inside a component will add a delay, and therefore the additional virtualization layer could potentially result in higher latency. To isolate the testing packages will only be sent though the same single switch as in the throughput tests. And the resulting delay will be the total end-to-end delay between the two systems. The only variable is the layers of virtualization at the host.
For the latency test the netperf tool is used, with the following command:

```
netperf -H HOST -t TCP_RR
```

The results for the two testing environments namely, HP and Dell, are indistinguishable. For that reason only the results for the Dell servers is presented. As discovered in the throughput tests the use of the virtio driver is essential for performance. Any differences in the latency are probably insignificant due to the poor overall network performance without paravirtualization. But benchmarking of both scenarios are carried out for comparison purposes.

First the case where only the default emulated network device is used, and no virtio driver, is looked at.

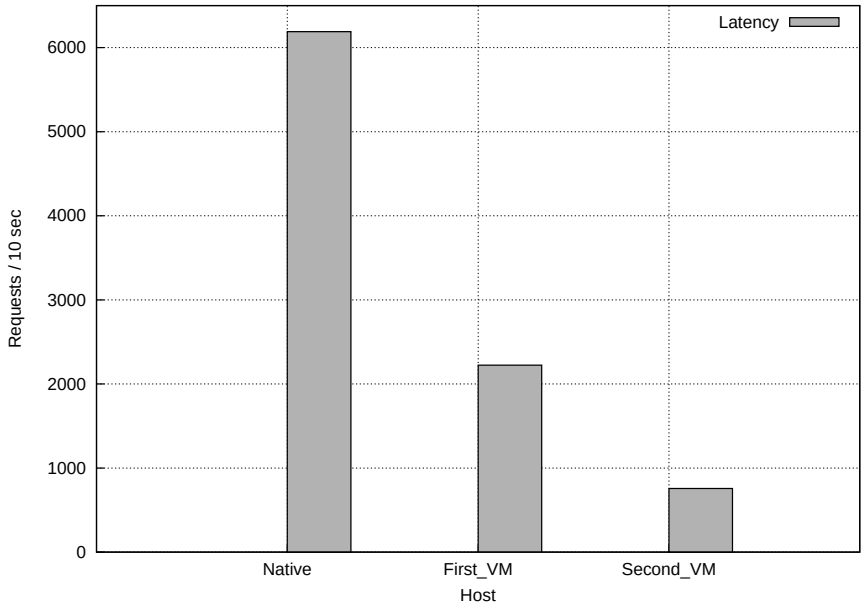

Figure 4.7: Network Latency on DELL test-rig

On figure 4.7 the y-axis represents how many connections the server could handle over a ten seconds period. In the case of the native server this is 6179.26 requests. When the test is run again towards the first virtual machine the request count drops to 2161.07 req/10sec. And with the second, nested, VM a further drop to 644.44 is measured. As expected this bears resemblance to the throughput tests where the performance drop was significant without the virtio driver. The first VM has only 34.9% of the native performance, and the second layer has only 29.8% of that again. This gives the second layer 10.4% of the native performance.

The virtio driver is then activated in both the first and second layer of virtualization and run the same tests again. Figure 4.8 shows the results.

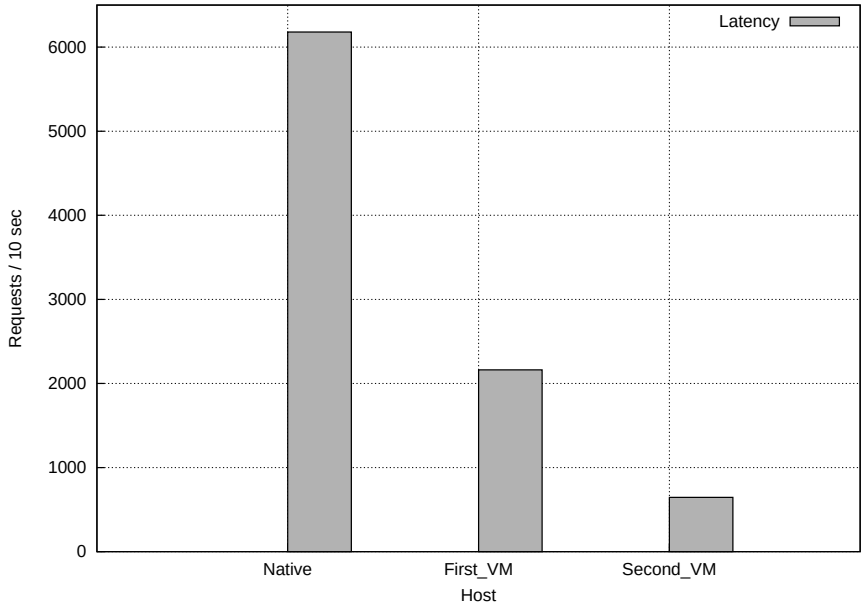

Figure 4.8: Network Latency on DELL test-rig with virtio driver

The native performance is similar to that of the previous test at 6189.48, as no changes has be made to the basic setup. But surprisingly there is little change in the responsiveness of the first or second layer VMs as well. The first VM scores 2223.39 requests per 10 seconds, which is 35.9% of the native result, and only an improvement of 1% compared to the non-paravirtualized network driver. The second layer manages 756.90 request and is 12.2% of the baseline value. For the nested virtual machine the paravirtualized virtio driver gives only an improved performance of 1.8%.

The finale latency test is a simple script executing ping requests over ICMP measuring the round trip time, RTT. This is the time it takes from the request is sent to the acknowledgement packet is received. This test was executed with the virtio driver in use.

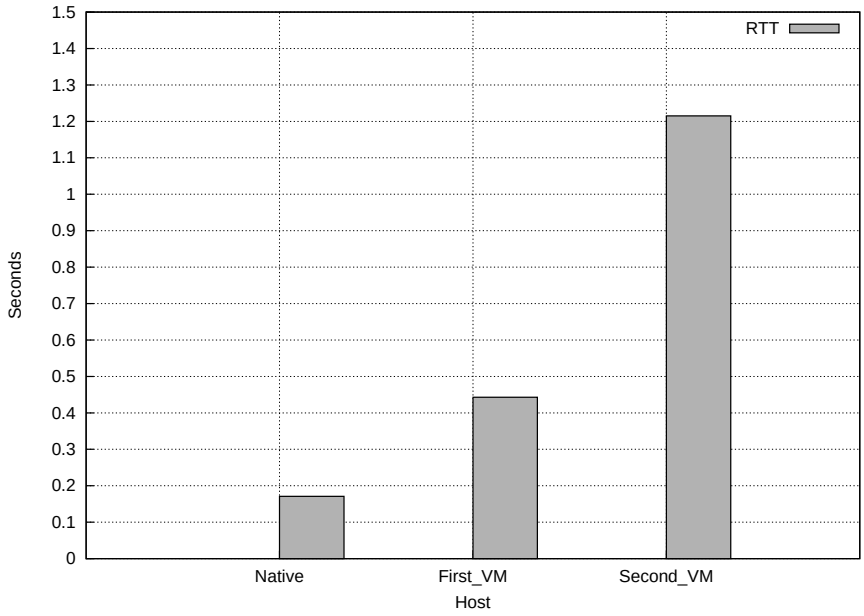

Figure 4.9: Network Round trip time on DELL test-rig with virtio driver

The native physical server has a mean round trip time of 0.171 milliseconds. The first layer VM clocks in at 0.443 ms, which is 61.3% slower. And the second layer takes 1.215 ms, 85.9% slower than the non-virtualized server. This is 38.0% and 14.0% of the native test respectively. The simpler round trip time test is therefore consistent with the netperf tests.

4.2.3 Disk

To measure the disk performance the *Bonnie++* test program was utilized. It was executed with the following command:

```
bonnie++ -d tmp/ -s 8192 -r 4096
```

The factors to concentrate on are sequential reading and sequential writing of blocks to and from the disk and also the random seek time. The sequential operations indicates the maximum read or write performance when eg. copying large files, while the random seek shows the rate of which small files can be accessed at random locations on the disk.
The performance values on the Intel and AMD systems where identical, so the focus is on the AMD powered Dell servers as with previous tests.

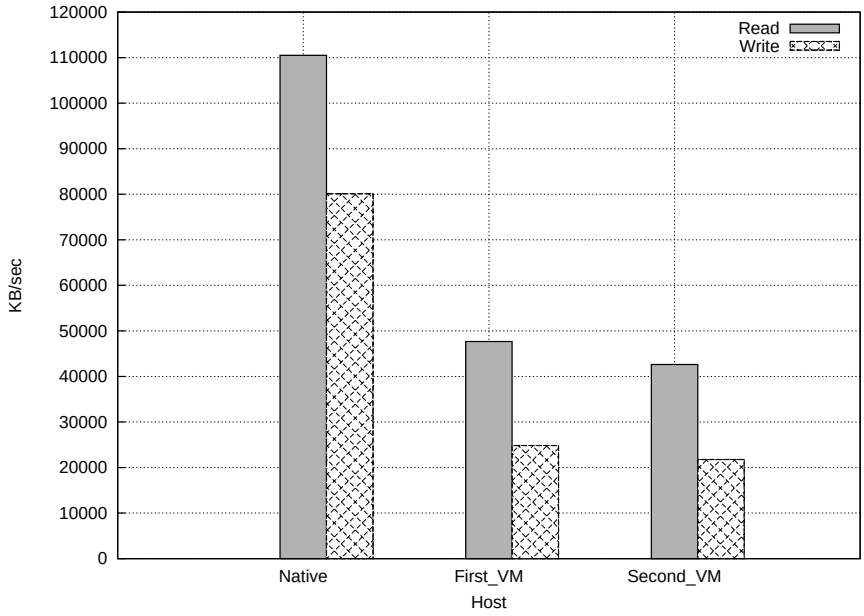

Figure 4.10: Sequential read and write

Figure 4.10 shows the read and write speed of all three layers of virtualization, all with default settings. The first bar is for the native server. Read performance is expected to be better than write performance at all levels. For the first layer of virtualization the performance has dropped significantly, the read is now 47.6 MB/s compared to 110.4 MB/S without virtualization. This is only 43.1% of the original server. The write speed suffers a little more at 31.0% of the native performance. For the second layer a further drop is seen. The read speed is now only 38.5% of the original and the write 27.1%. That is an additional drop 10.5% and 12.4% for read and write respectively added by the nested VM layer.

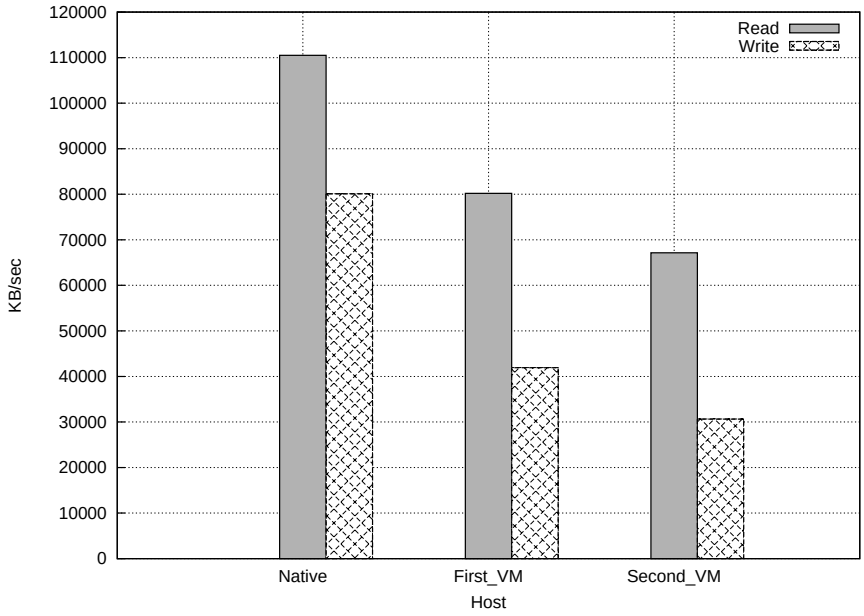

Figure 4.11: Sequential read and write with the virtio driver

Figure 4.11 shows the same tests as the figure 4.10 but here the paravirtualized virtio driver is used for both layers of virtual machines. The native servers has the same values, but the first and second layers of VMs are now closer to the original speed. The first layer VM has a read rate of 80.2 MB/s and write of 41.9 MB/s. That is 72.5% and 52.3% of the native performance and an improvement of 68.2% for read and 68.7% for write compared to the results without the virtio driver.

The second VM achieves a read at 67.1 MB/s and write at 30.6 MB/s. That is 16.2% for read and 26.9% for write down on the first VM, but it is also 57.4% for read and 40.0% for write up on performance compared to the second VM without the virtio driver.

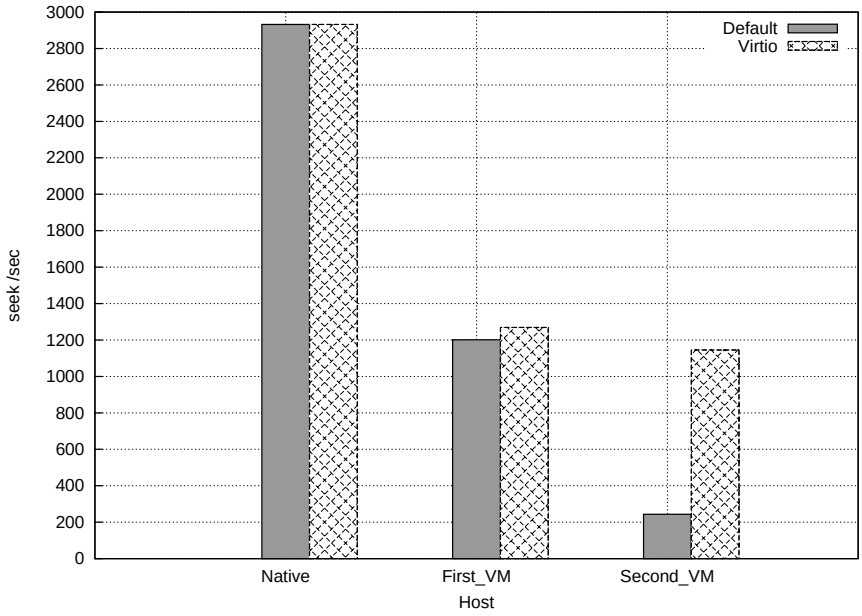

Figure 4.12: Random seek

The last test is the random seek disk test. The Y-axis is here the number of seeks completed in one second, and the higher the better. On the native server both results are the same since the virtio driver does not have any effect. For the first normal VM there is a very similar drop in performance for both the standard VM and the VM using the virtio driver. They are only 5.3% apart with the one using virtio in front. They are 59.0% and 56.7% behind the native server.

However, a look at the results for the second, nested, VM clearly shows the difference. Without virtio the number of seeks per seconds plummet to 243.3. That is an overhead of 91.7% compared to the native server. With the virtio driver it is only 9.7% fewer seeks than the first VM.

4.3 Live migration

Much of the motivation for running a cloud inside other clouds or on top of virtualized infrastructure were flexibility, and most of the added flexibility comes from being able to live migrate virtual machines between the different infrastructures. The tests are to see if the live migration works and what combinations of clouds and hypervisors that are required to support seamless migrations.

As discussed earlier, there are different layers to consider. Machines can migrate on two abstraction layers, the cloud node or the cloud guest. If we have a private cloud hosted on our own hardware and a second private cloud inside that one with guest running on it we have the three layers, L0, L1 and L2. When we own the whole stack we can migrate at two levels, the first being the L1 VMs. These are normal virtual machines when viewed from the hypervisor below, and can be live migrated as a normal virtual machine. But these VMs are cloud nodes themselves, with a nested hypervisor hosting additional VM inside them, the guest VMs. So when viewed from L1, the virtual cloud node, all guest VMs can be migrated between the cloud nodes in the virtual cloud.

The other possibility is when using a public cloud to host the first layer of virtual machines, L1. Now we do not have control over the first layer, and can therefore not configure shared storage or any other requirements for live migration. This means that live migration of the L1 cloud nodes will not be possible. An example is that we can not live migrate an EC2 instance around or out of the Amazon cloud. But when using a nested hypervisor we do have control over the third layer L2, so these guest VMs can be migrated on top of the L1 nodes.

Live migration require shared storage, but it also requires that both hosts, the sender and the receiver, have the same type and version of hypervisor. The version could in some cases vary slightly and still work, but this is often not guaranteed. Traditional live migration across heterogeneous hypervisors will not work. From a technical standpoint hypervisors are not fundamentally incapable of migration between each other, but this functionality is not implemented in any of the major hypervisors.
Below we will look at some example setups and show that live migrations are possible.

4.3.1 Private Cloud

The first setup is when layer L0 are a private cloud where one has control over the physical hardware and the hypervisor. The whole stack are now under the users control. One of the servers acts as the cloud front-end and cloud controller while all the other servers will be cloud nodes. The cloud controller will also be the central storage by exporting the folder where the virtual machines are stored with NFS. All the cloud nodes mount this NFS share and will therefore have access to the disks of all VMs without the need to copy them across the network. The nodes have all installed the KVM hypervisor. Which private cloud software in use does not rally impact the results as they just manage the nodes and VM and rely on the features of the underlying hypervisor, but for this test OpenNebula is used. This is now a normal private cloud and will act as the base for the new layer. Virtual machines can be created and booted on the physical nodes as normal, and then live migrated from one node to another as shown in figure 4.13.

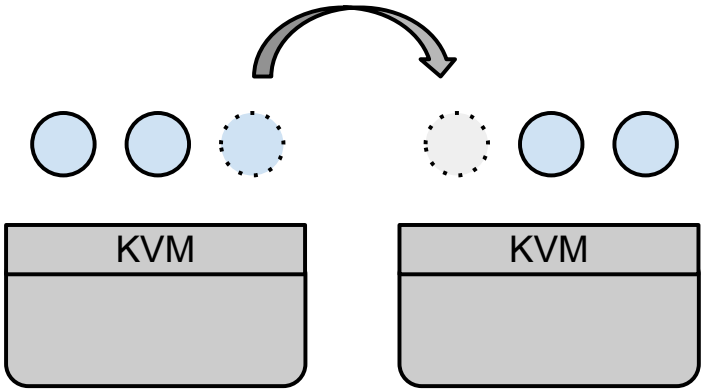

Figure 4.13: Live migration of L1 VMs

Now with a fully functional private cloud with live migration capability the second, nested, layer is added. The installation and setup is almost identical to the cloud already running. Instead of installing the cloud controller and cloud nodes on physical machines they are installed on the virtual machines now running in the existing private cloud. This requires a working installation of a nested hypervisor as proven above. The cloud controller for the L1 private cloud is installed in one of the virtual machines and will also act as a NFS share for its nodes. This L1 NFS share must not be confused with the L0 NFS share previously installed. The L1 nodes acts the same way as the L0 nodes and the KVM hypervisor is installed to enable VMs to run on top of it. With this setup complete L2 virtual machines can be started and running inside the L1 private cloud. Since the L1 cloud nodes also mounts a shared storage for the VM disks L2 guest VMs can be live migrated back and forth between the L1 nested cloud nodes. Figure 4.14 shows a virtual machine live migrating from one L1 cloud node to another, and in this case that also implies a migration from one L0 cloud node to another.

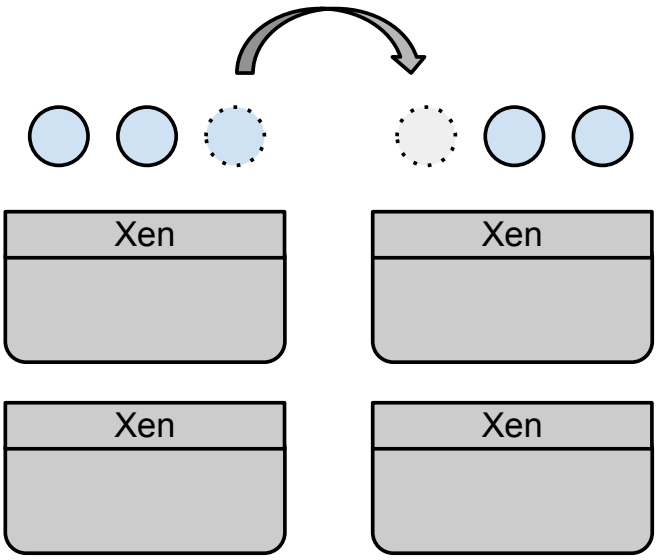

Figure 4.14: Live migration of L2 VMs

The setup and test described above depends on total control of all layers and homogeneous hypervisors inside each layer. The two private clouds, one inside the other, is however not dependent on each other. We can swap the L1 hypervisors for any other hypervisor suited for our needs. And similarly we can swap the L0 hypervisors as long as they support nested virtualization.

All major private cloud software solutions today supports a heterogeneous set of node hypervisors. That means that for example half of the L0 nodes run the Xen and the remaining half use KVM. For the user of such a cloud the experience will be the same with the exception that live migration of VMs could not be performed from one hypervisor type to the other. If this is the case it partially breaks the flexibility of live migrating the L1 cloud nodes. This limitation will only impact the L1 VMs and not the L2. Because L2 is independent of L0 we can still live migrate the L2 VMs across all the L1 nodes and that in turns means across all the L0 nodes, even though that means from KVM to Xen. This is illustrated in figure 4.15.

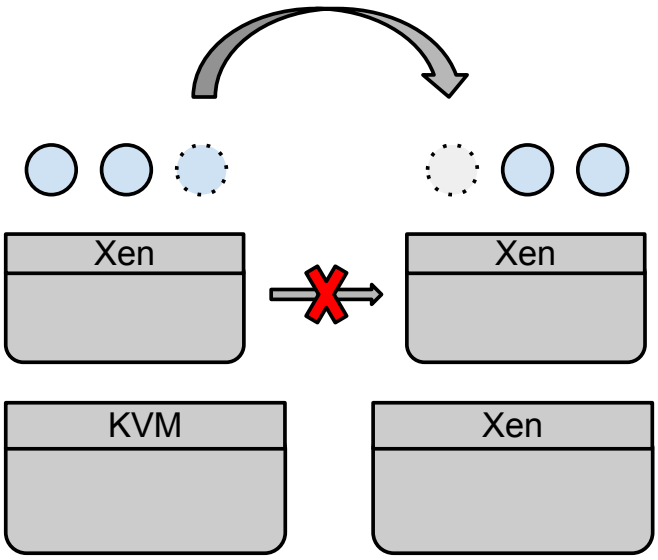

Figure 4.15: Live migration of L2 VMs

With the nested clouds setup it is also possible to live migrate the L1 virtual cloud nodes with the L2 cloud guests inside. For this to work the L0 hypervisors must be the same. This will enable us to migrate complete clouds from one location to another with all the L2 guests intact and running.
Figure 4.16 shows that if both L1 nodes is running on the same hypervisor we can live migrate that node and all guests running inside it.

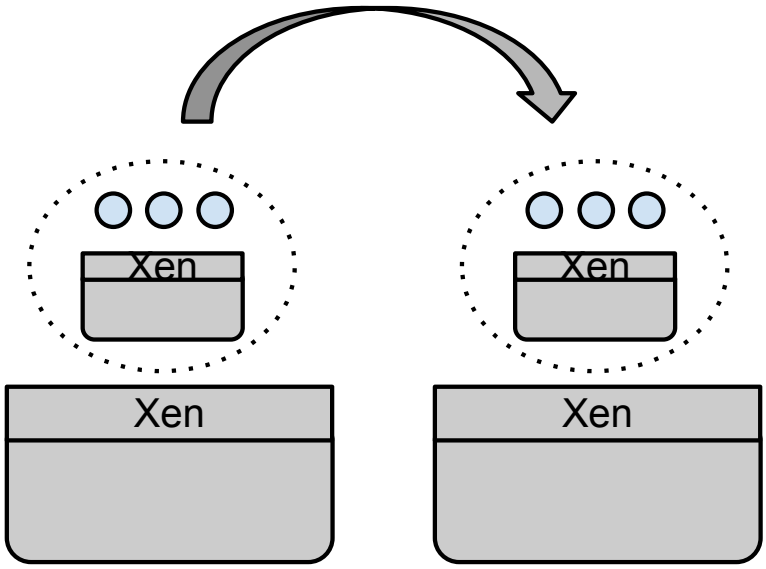

Figure 4.16: Live migration of L2 VMs

4.3.2 Public Cloud

We will not always be able to have full control over the whole stack in our nested cloud setup. By using a public cloud as our L0 we loose our ability to configure the L0 hypervisor and shared storage. This is the case when using a public cloud like Amazon EC2. It also applies to other public or private clouds not owned by the user. If we want to use a private cloud from another company to host our cloud we will not control the L0 even though it is run with a private cloud software like OpenStack and the Xen hypervisor. What software or hypervisor the L0 cloud uses does not matter, as long as it supports nested virtualization, we can still host our private cloud inside the unknown public cloud. This L1 private cloud gives us back the full control over the hypervisor and allows us to configure shared storage and the requirements for live migration. So it does not matter what system or hypervisor L0 has as long as it supports nested virtualization. Then the second layer will give us live migration across all L0 clouds.

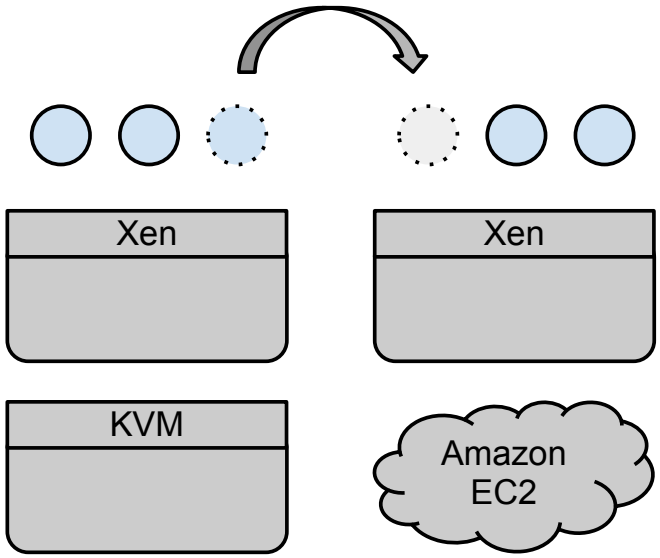

Figure 4.17: Live migration of L2 VMs to a public cloud

4.3.3 Cloud compatibility

The live migration tests carried out across a range of nested clouds boils down to a few simple patterns. These tests depend on tested software available to implementation today. First we look at the classic examples not using nested virtualization.

- No virtual machine can be live migrated from one hypervisor to a different one. That means a VM running on for example the Xen hypervisor can not be live migrated to a KVM platform or the other way around.

- Equally a VM can not be live migrated from one cloud platform to a different cloud platform using the standard cloud API. So a VM running in an OpenNebula cloud could not be migrated to a OpenStack cloud. This is only if we consider the standard APIs of the clouds. If the two clouds used the same hypervisor and we had total control over both, eg. root access, we could theoretically manually migrate the VMs using the hypervisor APIs. This is however not a practical solution as the VM moved to the other cloud would not be registered correctly and might break the clouds control over the VMs running on it.
- A virtual machine can not be live migrated to or from a cloud outside our control. That means that if the two clouds have the same software and use the same hypervisor it will still not be possible to migrate the VMs across if one of them was not in our control, eg. root access.
- Live migration of virtual machines to or from a public cloud and a private cloud or between two public clouds is not possible. This is based on the same principle as the previous point, i.e. the lack of control.
- The only fully supported live migration operation is between two homogeneous private cloud nodes. All the previous points means that to live migrate a VM between two cloud nodes they must have the same software, use the same hypervisor and be in our control.

After implementing the nested cloud concept we can do similar tests in order to see an improvement. The following points are the results of live migration test on the nested cloud layer. The nested cloud used here is OpenNebula with Xen as the L1 hypervisor. This require L0 hypervisors with support for nested virtualization.

- A nested guest, L2 VM, can be live migrated from one L0 hypervisor to a different one. The L2 VM does actually migrate from and to the same L1 hypervisor, but across different L0 hypervisors.
- If the L0 cloud is under our full control live migration of a whole L1 cloud node, with all its L2 guests, is possible.
- The nested guest can be migrated across different L0 cloud software. As with the first point this is because the L0 layer does not impact the live migration on the L1 cloud.
- It is not necessary to have control over the L0 cloud to live migrate L2 guests to and from it.
- It is possible to live migrate L2 virtual machines between a public and a private cloud.
- The above point means that fully heterogeneous live migration is possible if all the involving L0 hypervisors support nested virtualization.

Below we can see a summary of the migrations on a table form.

L0	Hypervisors		Migrate L1 VM	Migrate L2 VM
	From	To		
Private Cloud	L0 Xen	L0 Xen	√	√
	L0 Xen	L0 KVM	X	√
	L0 KVM	L0 Xen	X	√
	L0 KVM	L0 KVM	√	√
	L1 KVM	L1 Xen	-	X
Public Cloud	L0 Xen	L0 Xen	X	√
	L0 Xen	L0 KVM	X	√
	L0 KVM	L0 Xen	X	√
	L0 KVM	L0 KVM	X	√

Table 4.1: Table of working hypervisor combinations

In table 4.1 we can on the left side see the L0 clouds in use and the hypervisors its nodes are using. The cloud nodes are divided in two, the sender and the receiver. A private cloud is a cloud we have full control over, and the public cloud indicates a cloud where we can run VMs, but does not have privileged access to. The two columns to the right shows what actions we tried. Migrate L1 VM means that we want to live migrate a normal L1 VM from one cloud node to the other. The Migrate L2 VM is to live migrate a nested L2 guest VM running on the nested cloud inside the standard L0 cloud.

The first example is to live migrate a VM from one Xen cloud node to another also running Xen. This works for both the normal and the nested VM. The second test is to live migrate a VM from a Xen node to a KVM node. This will not work for the standard L1 VM as shown above, but for the L2 VM it works since it runs inside the nested cloud.

L0	Clouds		Migrate L1 VM	Migrate L2 VM
	From	To		
Private Cloud	L0 OpenStack	L0 OpenNebula	X	√
	L0 OpenStack	L0 OpenStack	√	√
Public Cloud	L0 Amazon EC2	L0 Amazon EC2	X	√
	L0 Amazon EC2	L0 OpenNebula	X	√
	L0 Amazon EC2	L0 OpenStack	X	√

Table 4.2: Table of working cloud platform combinations

Table 4.2 is on the same form as 4.1 and shows how the cloud software not can migrate VMs between each other on the first, L1, layer. But on the nested second VM layer they can. This also includes public clouds like Amazon EC2.

Most public clouds does not yet have support for nested virtualization. This is probably because the technology is too young and immature to be implemented in production environments or not desirable for the provider to imple-

ment. In such cases full hardware virtualization on L1 will not be possible, for example KVM. But paravirtualization with Xen is still technically achievable through relatively minor changes to the Xen and Linux drivers. This problem is in research and development stages under the *Xen-Blanket*[38] project. Their goals are to implement a second layer Xen hypervisor on top of non nested-enabled hypervisors, like Amazon EC2, with no or minimal performance loss compared to normal nested paravirtualization. It is this technique we used in the Amazon migration tests.

Chapter 5

Analysis and Discussion

In this chapter the data obtained in the previous chapter will be analyzed. The results will first be analysed statistically to ensure its validity and significance. Later, the findings will be discussed and their meaning explained. This will end up in a discussion of the concepts feasibility and how the principles could impact the marked and future implementations of clouds environments spanning multiple virtual infrastructures. Future research will also be suggested and discussed.

5.1 Synthetic benchmarks

The results obtained in the synthetic benchmarks will first be describe and analyzed. This is primarily to find the differences between one and two layers of nested clouds and if there is a significant performance loss or change in the behavior. We will start with the processor tests on both the HP and Dell systems.

5.1.1 Processor

The first test was on the HP system with an Intel CPU. From the figure 4.1 displayed in the result chapter a significant difference was observed when introducing virtualization to a native setup. We would expect the native operating system to have the best performance and the highest consistency. And with an average, or arithmetic mean, of 3.56 seconds per calculation it is considerably faster than the virtual machines. The consistency is also great with a standard deviation of just 0.0131 over a 200 sample test. The distribution is shown graphically in figure 5.1, and is close to a perfect normal distribution and with a very small spread of just 0.08 seconds, and these are all indications of a consistent and accurate level of performance.

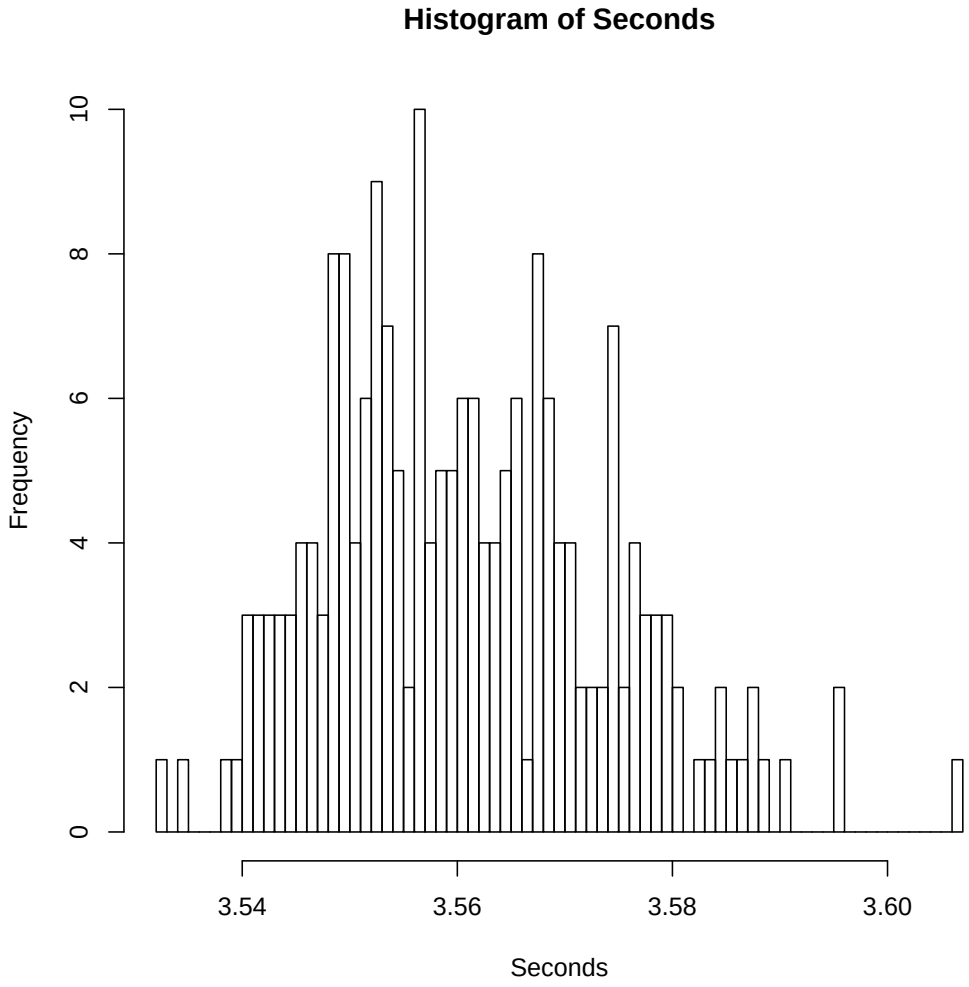

Figure 5.1: Distribution L0 hp

The next test was on the same server, but inside a virtual machine in a cloud. The average for all 200 samples where 4.35 seconds. That is 18.1% behind the native performance, which is expected as virtualization always will introduce an overhead. However, it is not expect that the standard deviation is lower than the native server at 0.0018. And the sample spread is even closer with just 0.015 seconds from the lowest to the highest. When introduce a virtual layer one would expect lower performance and slightly more inconsistency and higher standard deviation, but in this case the performance is poorer and the samples are closer together. The graph 5.2 shows the distribution of the samples and resembles a long tailed Poisson distribution. The long tail of the distribution is at the higher end of the graph and shows that some of the samples are higher, but very few are lower, than the average. This is an indication of a fixed overhead in the virtualization where no sample is below, but most are close. There are outliers and some variation, but these are above

the fixed overhead and higher than the average, which gives us the long tail. Normally the lower spread and standard deviation would be surprising when the performance is lower, but when considering the fixed overhead it makes sense as this overhead cuts off the natural spread at a very precise point.

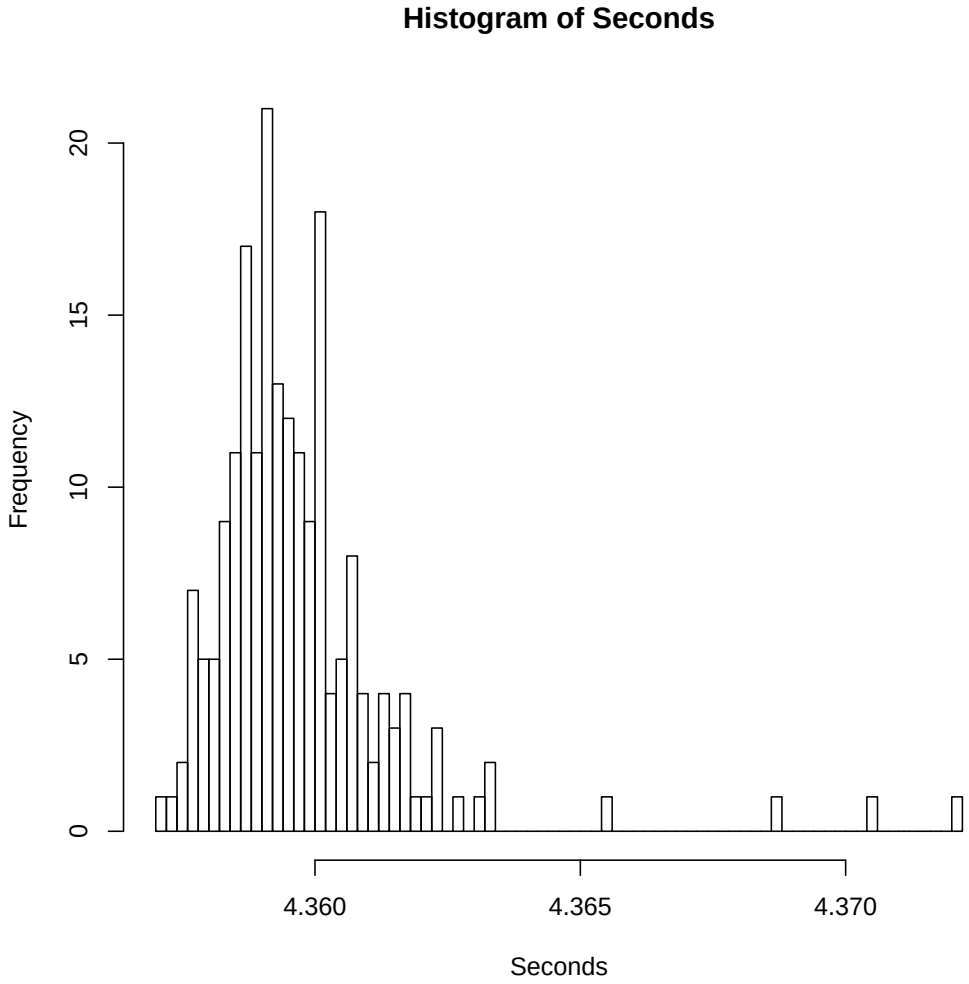

Figure 5.2: Distribution L1 hp

The third and last test on the HP test-rig is the nested cloud VM. The virtual machine running these tests where hosted inside the VM described in the previous test. As this is an additional layer of virtualization the result is, as expected, even lower performance. The drop is however not as great as from the native server to the first layer of virtualization. The first layer had 18.1% lower average, and the second layer is only an additional 6.6%. An average of 4.66 seconds, 23.6% behind the non-virtualized server. This means that the second layer cloud does not have en equal overhead as the first layer. The standard deviation is now 0.0029 and the distribution is shown in graph 5.3. That is

much closer to the first layer of virtualization than the native server and that shows in the distribution graph as well. It is the same type as the first layer but with more spread and a longer tail. The tail is still to the right, indicating that some of the samples are slower, but no one faster, than the cut off similar to what we saw in the previous test. Overall the results are comparable to the L1 cloud and the behavior is equal, but the performance is slightly behind, as would be expect with the added layer overhead.

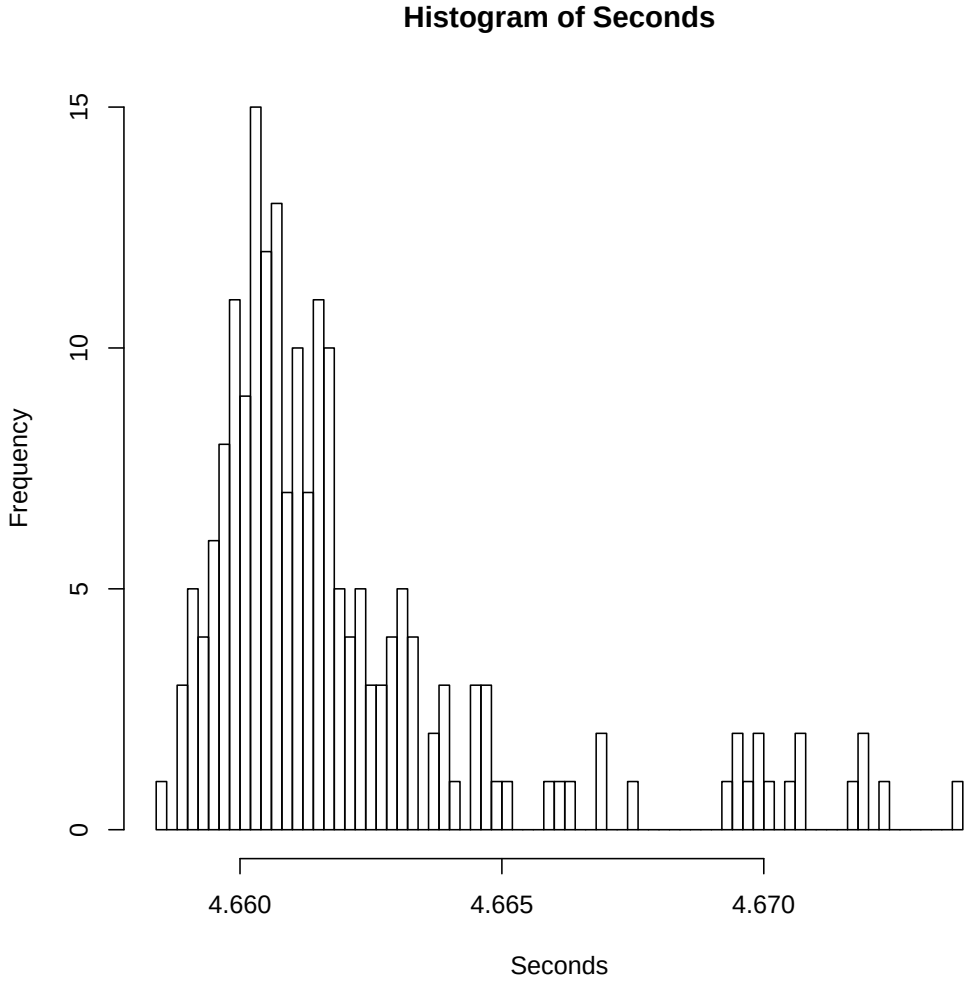

Figure 5.3: Distribution L2 hp

The same three tests where executed on the second test-rig, the AMD powered Dell servers. The first was the native baseline test. At 4.76 seconds on average it is considerably slower than the Intel servers, but that is just because the different processors are insignificant for the results. The standard deviation is 0.01, which is also much poorer than the HP systems, and again not the subject for this thesis. What we can observe from the distribution shown

in graph 5.4 is that it has close to normally distributed behavior around the average with a few outliers.

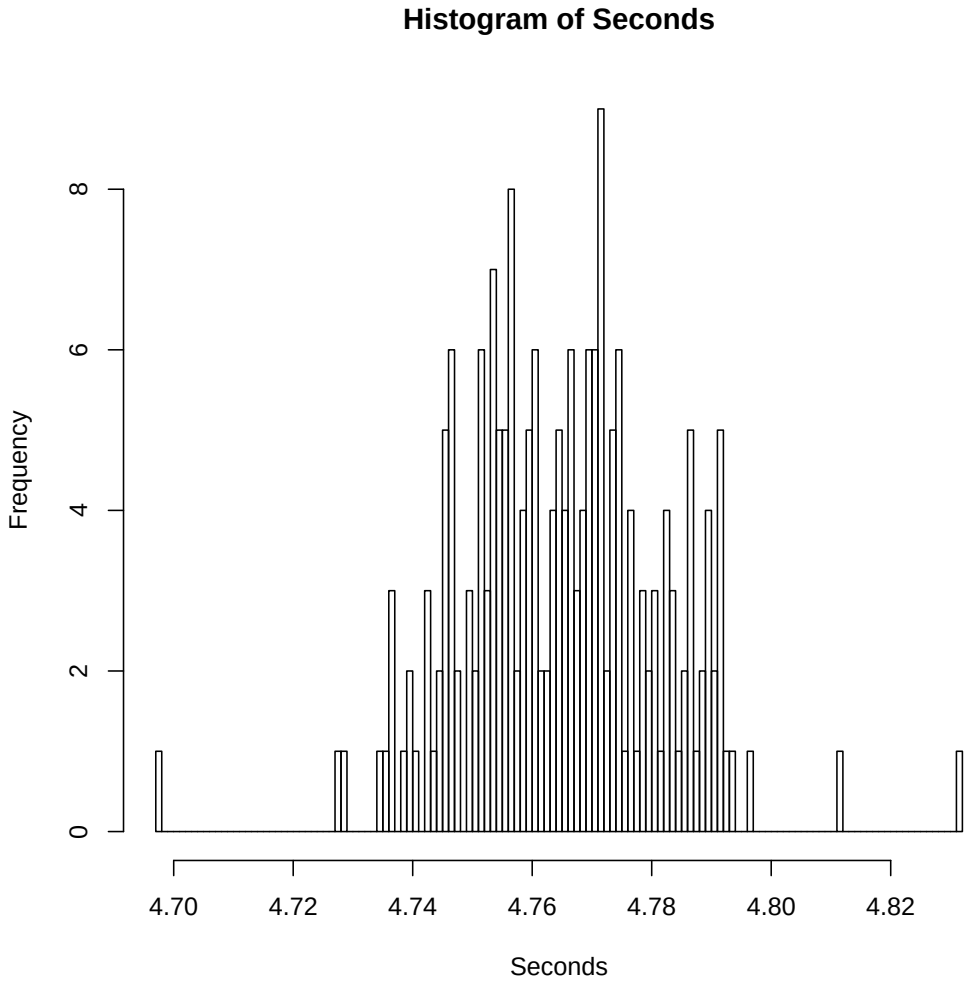

Figure 5.4: Distribution L0

As noticed in the result chapter, the first layer of virtualization on the AMD system is surprising in the way that it is faster than the native server. This should not be possible whatever the hypervisor or technology. It could be explained by software and a more optimal algorithm, but the operating system, kernel and all software are identical on the physical server and the virtual machine. There are also no restrictions or different prioritizing of resources between the VM and native server. The only relevant difference is the signature of the processor. The actual CPU in the physical server is a *Quad-Core AMD Opteron(tm) Processor 2376* as reported by *lshw*. The CPU represented to the virtual machine by the hypervisor is reported by *lshw* in the VM as *AMD Phe-*

nom(tm) 9550 Quad-Core Processor. One possible explanation for the better VM performance is that the Linux kernel detects the CPU type and adjust some settings for optimal performance, eg. the scheduler. Similar phenomenon has been observed on AMD systems previously[21]. But even though the results are surprising it does not directly affect the work in this thesis. We are mainly interested in the difference between the first and second layer cloud VMs. Normally both layers of virtualization layers would be slower than the native server, but this loss is already widely accepted in the industry as virtualization is common.

The distribution of the first layer of virtualization is shown in figure 5.5. The spread is wider than the physical machine with a mean of 4.57 seconds and a standard deviation of 0.04.

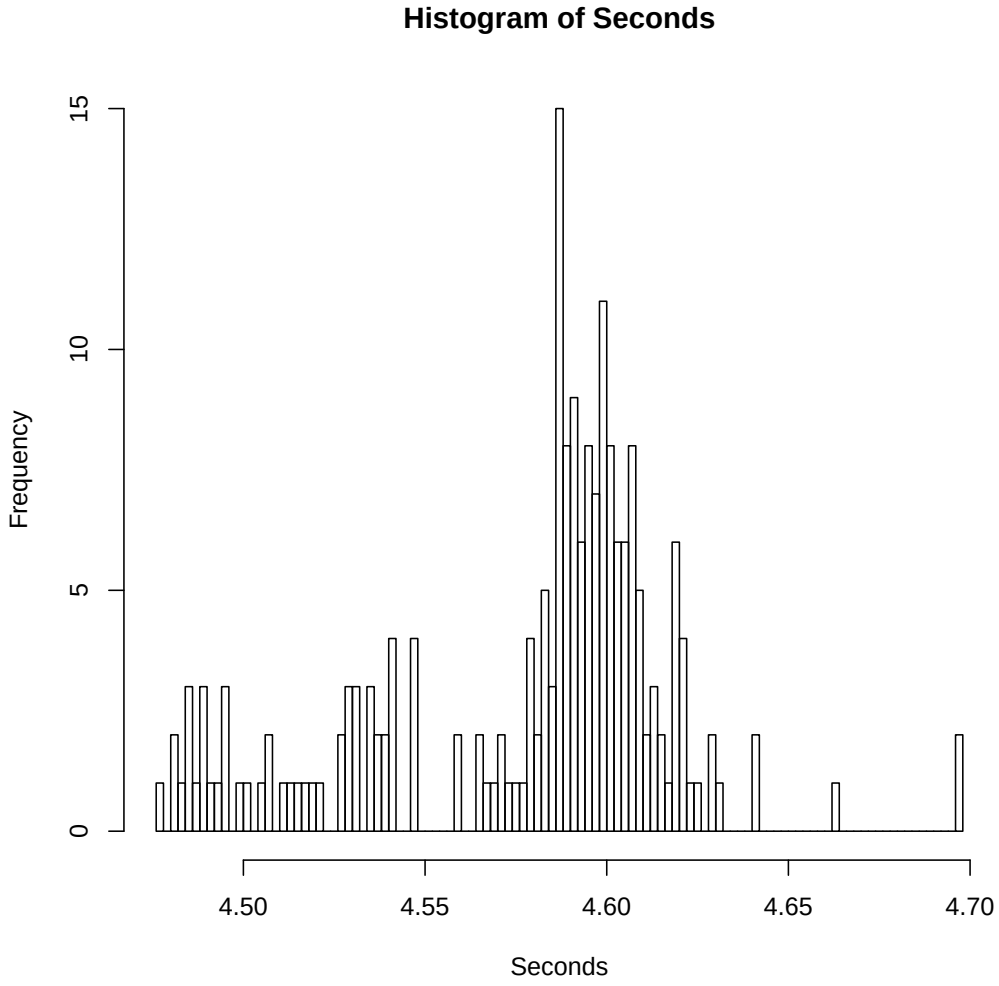

Figure 5.5: Distribution L1

The third test on the Dell servers where the second virtual cloud layer. The

mean is now 4.80 second with a standard deviation of 0.10. Figure 5.5 shows the distribution. With the AMD Dell servers we do not see the long tail to the right as the Intel HP servers. Now we see a long tail to the left. This indicates that on some occasions the results are better than the mean. The best samples from the second layer VM, the extreme right outliers, are in fact better than the mean of the first layer VM.

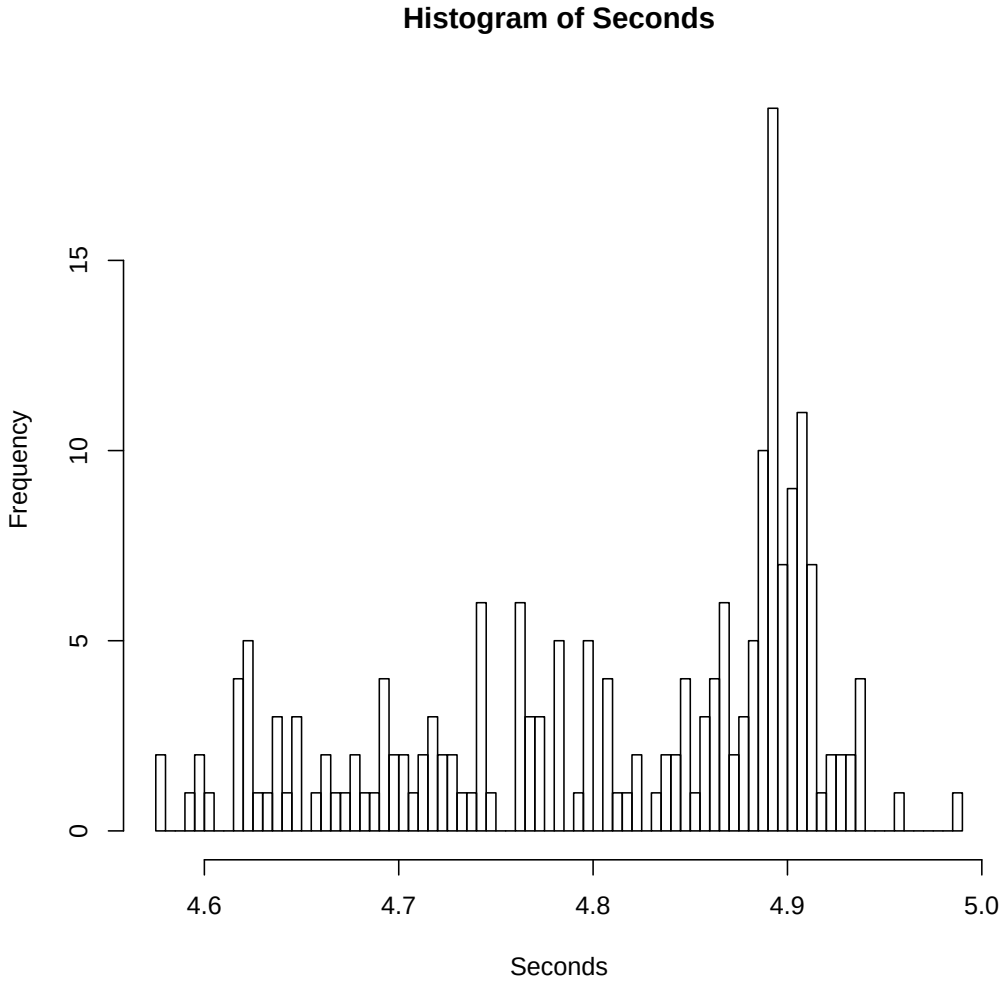

Figure 5.6: Distribution L2

5.1.2 Network

The network performance is important for how the total system functions, and can be divided into two main categories; throughput and latency. Throughput is how fast data can be transfered across the network and latency is the time one connection takes. Both factors were measured on the two test systems to find the overhead of running one cloud nested. As with the processor test a

small overhead would be expected with each layer of virtualization, and the objective is to examine the additional loss in performance the second layer represents. If the difference between one and two layers of cloud is at an acceptable low level it can be justified by the extra flexibility that is gained.

Throughput

The first test on the HP servers gives us the baseline for the network performance, 934.04 Mbit/s without virtualization in a 1 Gbit/s network is the maximum possible transfer speed with the current equipment. The result from the second test is therefore surprisingly poor. The same amount of date can only be transfered at 242.46 Mbit/s when tested from the first layer VM. That is a drop of 74%, and not acceptable for production. The reason for this hugely disappointing performance is that the current implementation of hardware virtualization from Intel does not support IO devices. The network controller is therefore emulated and as a result suffers greatly. The second layer cloud VM has similar problems, and as it also inherits the poor performance from the first layer it has a network throughput of 35.02 Mbit/s. That is not considered usable for any kind of service and would halt any further testing with nested clouds.

Since this is a fundamental problem in the hypervisor and hardware virtualization and will not be solved before a third generation of x86 virtualization are presented there exists alternatives and workarounds. Partial paravirtualization of the IO devices with the *virtio* driver improves the performance. The native performance is of course unaffected and the same as earlier, but the first and second layer of cloud VMs are significantly improved. They have now 876.62 Mbit/s and 858.36 Mbit/s respectively which is only a 6.1% and 8.1% behind the native speed. For the experiments in this paper the most significant figure here is the additional overhead on the second, nested, cloud. Which in this case is as small as 2.0%. The tests where to transfer large amount of data over the network and was repeated 30 times. The results are the averages of these measurements. An interesting observation of the samples are the spread and standard deviations. The native servers have a standard deviation of 7.23, and the first virtual layer has 26.63. This means the virtualization introduces more inconsistency and the samples vary more even though the average is only 6.4% behind. The second, nested, cloud layer is a further 2.1% behind the first layer in performance, but the standard deviation is very close at 28.73. Comparing the three standard deviations we get an overhead of 72.8% between the native servers and the first virtual machine, and 74.8% from the native to the second VM. That is an additional 7.3% between the first and second layer of cloud VMs. This indicates that the major performance and consistency hit is introduced by the first layer. Any nesting on top of this has a much smaller impact on the network throughput.

The results where overall very similar for the Dell servers. All results are slightly higher, but this is probably down to the different network controllers.

The first layer of virtualization is 5.9% behind the native server, and the second layer cloud VM is 7.7% down on speed. This is similar to the HP servers, and also the difference between the two virtualization layers are nearly identical at 1.9%. The difference in standard deviation of the two virtual layers is also identical to the HP setup, at 7.3%. It indicates that the performance and consistency differences of the HP and Dell servers are non-existent beyond the hardware.

So even though the nested cloud VMs on the Dell servers were 4.2% faster than the nested VMs running on HP machines, their behavior is almost indistinguishable to one other and are well within each others standard deviation.

Latency

The second network test is to measure latency. There are two ways to find the latency in a system, one-way or round-trip. The first test were one-way, using the *netperf* tool. We started with a standard setup without the use of the virtio driver. This resulted in a large difference between the native server and the two virtual machines. The first layer VM has a performance drop of 65.0%. The second layer added an additional 70.1% overhead. This means a total drop of 89.5% from 6179.26 requests per ten seconds to 644.4 requests. This is very similar to earlier observations when the paravirtual driver is not in use.

By activating the virtio driver a more surprising result is observed, as we see almost no improvement at all. Previously this has given a significant performance boost, but now we can observe only 1% better with the first VM and 1.8% increase in requests at the second VM. These were alarming results, and raised the question of a fault in the virtio driver. The tests were repeated with identical outcome.

The second latency test is measuring the round-trip time of packets. A single ICMP ping packet is sent to the three different layers and the time it takes for the response to arrive back is recorded. The virtio driver is active in all tests. The result showed the same tendency as the one-way results, as there is a significant loss between each layer. The following graphs shows the distribution of round-trip times for the three layers. The first, 5.7 is the native setup and indicate great consistency with a standard deviation of just 0.003 and a mean of 0.17 seconds. The distribution is long-tailed to the right, higher values, which shows that we have found a consistent and precise baseline performance.

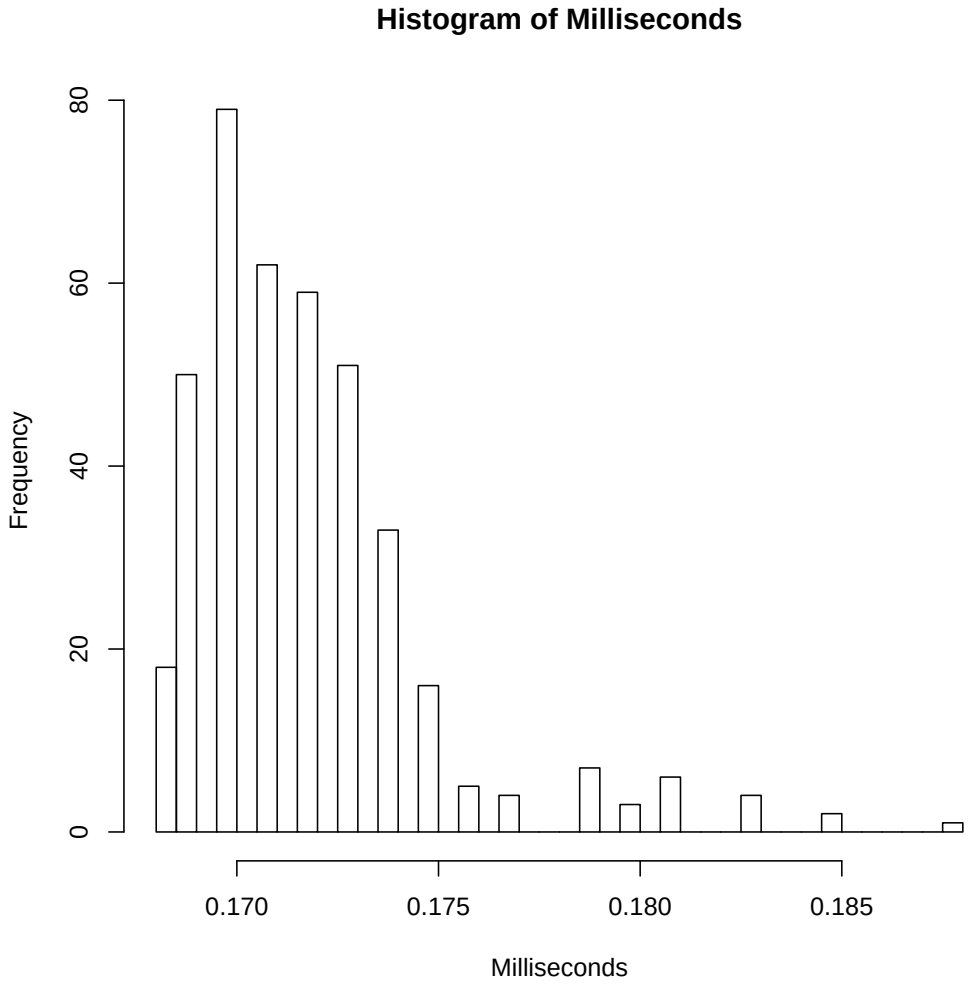

Figure 5.7: Ping Distribution L0

For the first layer of virtualization, figure 5.8, we get a mean of 0.44 seconds. This is only 38.8% of the native performance and as earlier proves that the virtio driver does not significantly affect the latency of virtual machines. The distribution confirms the credibility of the results by being close to normally distributed. The standard deviation is now 0.022, and shows a slightly wider spread of numbers.

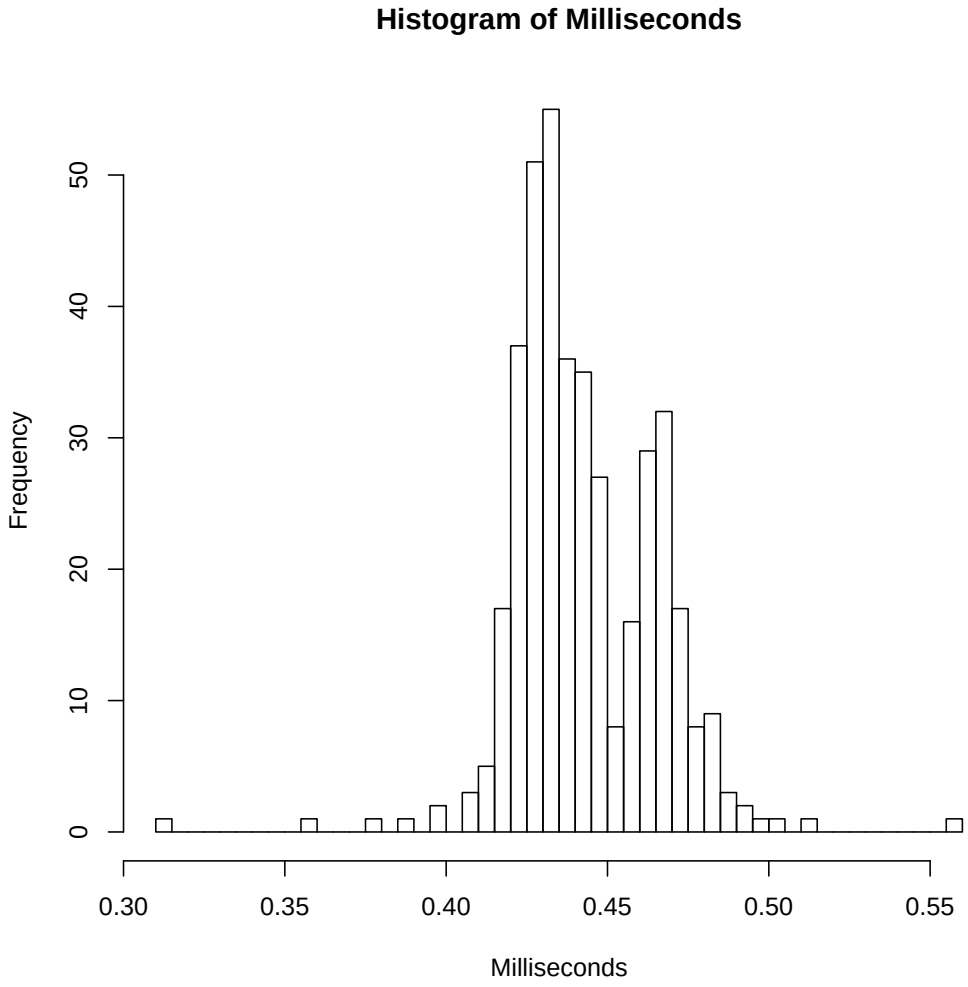

Figure 5.8: Ping Distribution L1

The trend continues for the nested cloud VM. The latency increases again with an additional 36.3% compared to the first VM. This is very similar to the overhead of the first VM, 38.8%. The standard deviation is now 0.074 and the distribution is a slightly long-tailed distribution.

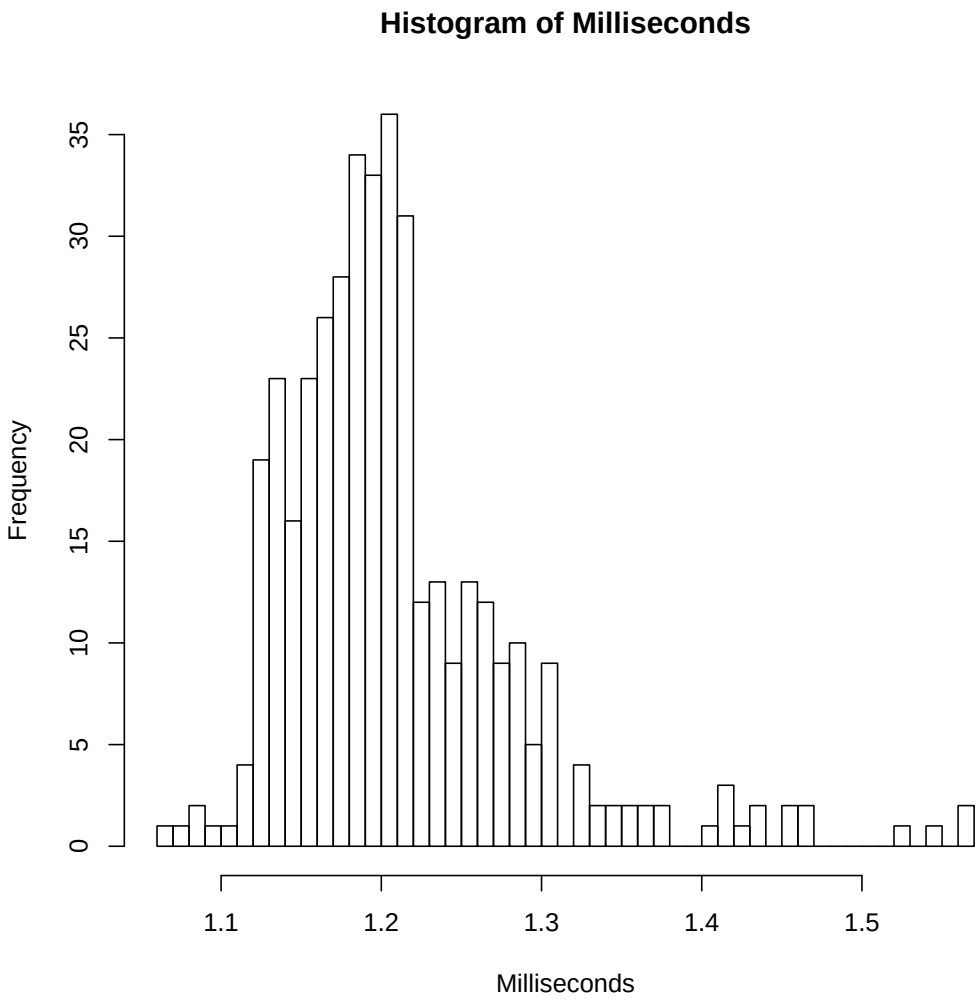

Figure 5.9: Ping Distribution L2

By looking at the standard deviation for the three distributions we can see a growing spread. This means that each layer not only adds a performance overhead, but also affects the consistency. The first layer has a 13.6% higher standard deviation than the native server, and the second VM an additional 29.7% higher SD. So the spread of the measurements increases more than the performance overhead introduced for each layer.

To further investigate the performance overhead in latency we run the same round-trip time test against a native server and forward the packages to a second native server. There is no virtual machines involved and the middle servers receives the requests through one network card and forward through a different one onto a different network switch. The three servers and switches are identical both in hardware and software. The round-trip times when measuring towards only the first servers is known from the first test, a mean of 0.171 seconds. The next test is to a third servers through the second, this way we find the overhead of forwarding packages through a server. The mean

this time is 0.429 seconds. This is nearly identical to the first layer of virtual machine. The standard deviation is 0.058 where it was 0.022 for the virtual machine. This could indicate that the latency overhead we see in virtual machines, regardless of the virtio driver, is due to general network overhead in the operating system. The spread and consistency in the virtual machine is tighter and better than by forwarding over physical servers and networks, likely because the virtual machines use software bridges which are more constant than real switches.

5.1.3 Disk

To test the disk performance a series of *Bonnie++* test where executed and get a general overview of the loss suffered with two layers of cloud. The first disk-test was a comparison of sequential read and write performance on all three layers. The native servers gives the true disk speed and each virtual layer had lower performance, as expected. Without the virtio driver this overhead for the first layer was 56.8% for read and 68.9% for write. For the second layer there is an additional overhead of 10.5% for write and 12.4% for read. These results indicates a relatively large performance hit when virtualizing, but the difference between the first and second layer is much smaller.

The same tests with the virtio driver gives a 27.4% loss in read and 47.6% loss in write speed on the first layer VM compared to the native server. This is considerably better than before, and the second layer of cloud VM is an additional 16.2% read and 26.8% write. Overall, the performance is significantly higher at all levels with the paravirtualized driver, but without virtio the performance drops around 60% from the native server, but the two virtual layers are much closer at around 11%. With the extra driver the speed is higher, but the difference between the first and second layer is bigger at around 20%.

The random seek tests also sees a significant drop when tested from the virtual machines. The non-paravirtualized scenario suffers greatly at both layers with 59.0% at L1 and 91.7% at the L2 VM, which is an unacceptable performance loss. With the driver the L1 is close to equal to the non-virtio result at 56.7%, but the L2 result is a much more positive 9.7% behind the L1 VM.

There is however great potential to improve the disk performance in a nested cloud setup. All the tests conducted in this thesis was done with virtualized storage. This means that the storage performance suffers the virtualization overhead twice, as the access to the disks goes through both the nested VM and the VM hosting the storage. This setup could be improved with physical storage solutions like SANs. To move in data-storage out of the virtual environment will improve speed but compromise flexibility, however due to the large amount of data it will often be much more static and not migrate around as the virtual machines. The network is a further limitation for the disk performance. All disk operations goes over the network as the VMs are hosted on the virtual cloud nodes and the storage on the virtual cloud controller. The net-

work speed will therefore play a vital role in the disk speed, and the network overhead of nested VMs is included in the disk overhead. The disk discussed here is where the operating system is installed, the root mount. Most applications running on top of a cluster of cloud VMs will have the content collected from a different source than where the OS is installed. For example a web server will get the content from a database or a shared storage that can serve all the nodes of the web service. This means that disk operations to the operating system disk is not as critical for the execution of the application. The logs will also in most cases be collected centrally in a monitoring system and not saved to the local disk. So after the initial boot of the virtual machine the application will execute from memory and get the content from external sources making access the VM image mostly insignificant. Further storage challenges are outside the scope of this thesis and requires much more planing and testing.

5.1.4 Performance summary

After testing and analyzing the performance and behavior of nested virtual machines in a private cloud we get a clear picture of the overhead and consistency. Virtualization will always impose some kind of overhead, and even when only one virtual machine is allocated all the physical resources of a server will be outperformed by the native system. This overhead is well known and documented[21], and the industry has accepted that virtualization has a small performance loss. The wide-spread adaptation [34] of virtualized servers and platforms shows that this loss in ultimate performance is in many cases an acceptable trade-off for the gained flexibility that virtualization represents. The purpose of the tests in this thesis were to examine the additional overhead of a second, nested, layer of cloud. The aim was to minimize the additional overhead compared to the first layer. If this extra overhead is sufficiently low it will be an acceptable loss for the gained flexibility. The loss should be considerably less than that of the first layer compared to the native performance. If the loss had been equal to the first layer loss it would bring the total loss up to a major factor and required a new assessment of the compromises between flexibility and raw performance. But on the other hand a low additional loss for the second layer compared to the first VM will encourage wider adaptation without a major further compromise.

The overall performance of the nested cloud layer is very encouraging. It is generally close to the first layer of virtualization. Figure 5.10 shows a summary of the performance of nested hardware virtualization, and is illustrated as percentage of the native performance. The most notable results are the good performance in the processor tests. This gives a general indication of the execution of virtual machines, and as with all the factors the important difference is the delta between the first and second virtual layer. The processor has an additional overhead of approximately 5% when nested inside a cloud.

The only unexplained phenomenon is the strange behaviour of the AMD pow-

ered Dell servers shown in figure 4.2. This behaviour has also been experienced by other independent benchmarks[21].

The network is not fully hardware virtualized, as these IO devices are not yet supported, and therefore a paravirtual driver is used to achieve acceptable results. The network throughput overhead is as low as 2%. The latency suffers more, but that is valid not only for the second layer of nesting but virtualizing in general. However, as further investigations of the behavior reveals, this overhead is nearly identical to forwarding the traffic through a server in the middle. And as the ratio between the native, first and second VM is similar to the other tests it is likely that the additional nested overhead will be minimized significantly if the latency on the normal virtualization layer is reduced. The disk tests has again the same pattern as the other results, with the nested VMs close to the normal VMs. The read is about 16% slower, the write 26% slower and the random seek only 9% behind the first layer. As previously discussed the performance of the disk operations is greatly affected by the network and the fact that the storage is also virtualized. This combined with the fact that VM image access often is mostly irrelevant after boot makes the disk performance much less important than the computation performance of the virtual machine.

Layer	CPU[1]	Network		Disk		
		Throughput[2]	Latency[3]	Read[4]	Write[5]	Random Seek[6]
L0	3.5279	933.27	6179.26	110495	80087	2932
L1	4.3813	876.62	2161.07	80202	41916	1269
L2	4.6327	858.36	644.44	67140	30641	1145

Table 5.1: Summary of the performance with virtio

Table 5.1 shows a summary of the results. The units are as following:
1: seconds.
2: Mbit/s.
3: req/10sec.
4: KB/Sec.
5: KB/Sec.
6: seeks/sec.

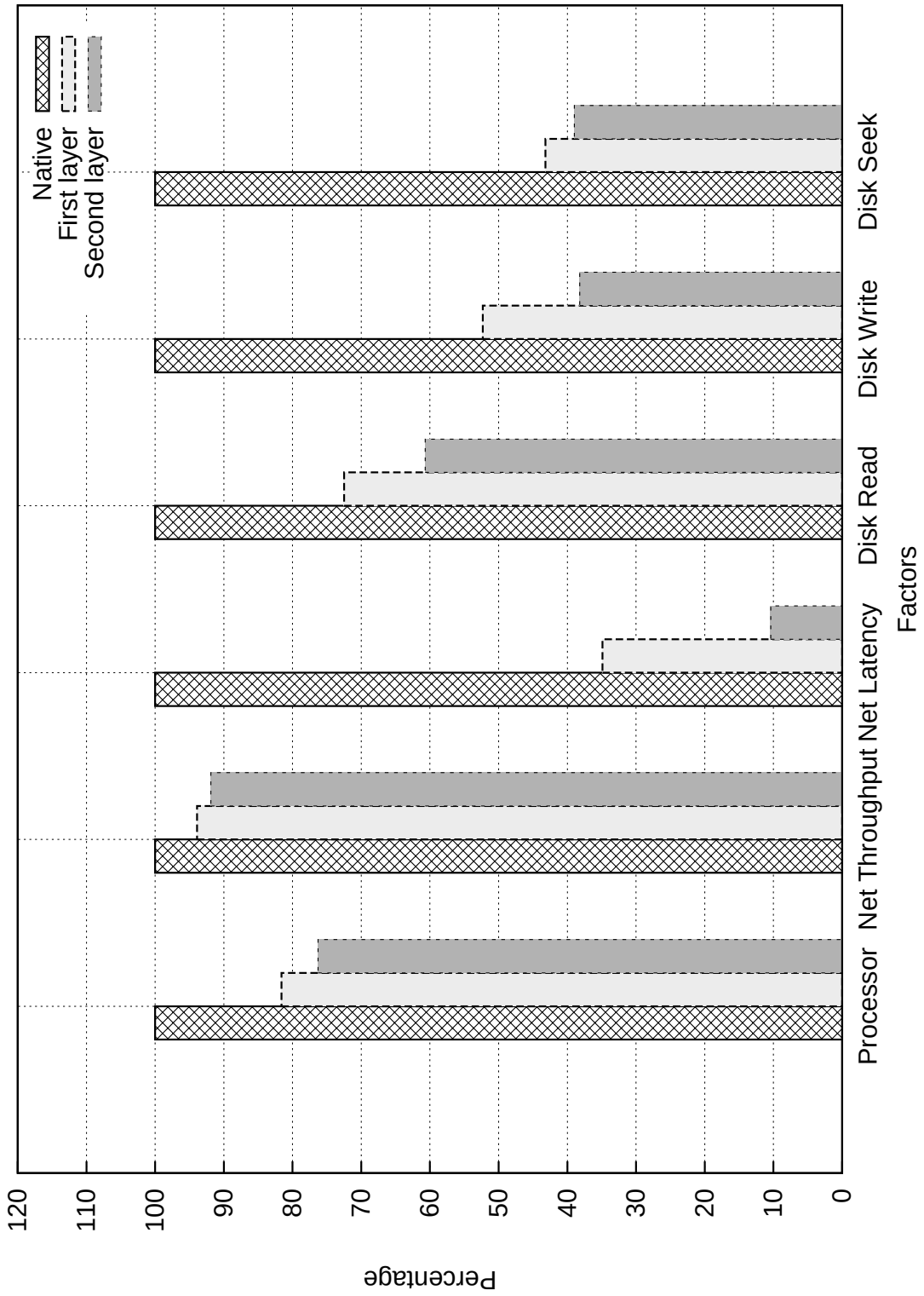

Figure 5.10: Summary of the performance overhead

5.2 Migration and flexibility

The synthetic benchmarks showed promising potential for running nested clouds. This opened the door for further tests and experimenting with the flexibility and added functionality that the nesting will bring with it. We will now look at the results and analyze the findings when hosting cloud guests inside a virtual cloud.

5.2.1 Nested clouds

Through all our experiments and testing we can clearly see that hosting one virtual private cloud inside another traditional private cloud does work. Our results show that a performance overhead is present, but at a low enough level so that the gained flexibility might be preferred over the small loss of performance. This will depend on the required specifications of the service to be hosted in the cloud. If maximum performance is the ultimate goal, nesting will not be a good solution. But for most modern services scalability and flexibility will have much higher priorities, and nesting will then be a lucrative alternative. As the results show, the difference between nested virtualization and standard virtualization is much smaller than the gap from running native to the first layer of virtualization. So the additional overhead is small if the service is already running in an virtualized environment. For a load balanced service, eg. web-service, the extra overhead could result in that one more node is required to maintain overall performance. This will mean higher cost if it is hosted in a public cloud or more physical resource-use is hosted on a privately owned cloud. But this extra cost might be saved by the flexibility to migrate to a cheaper provider or a more energy efficient data center.

Nested clouds are possible, but not all combinations or architectures will work. Depending on whether one has control over the whole stack of layers or only partial control will greatly dictate what solutions are possible to implement. Full control means full freedom to choose technology, and all options are therefore open. Only partial control will limit the options of implementation and flexibility might suffer. The two main levels of control are the two cloud layers. We will always have control over the second, nested, layer since it is based on the virtual machines in the first layer. But whether or not we control the first layer depends on the physical machines. If we own the physical infrastructure with servers and network we have full control. But often this layer is out of reach at a third party data center, either as a public cloud or as a private cloud administrated by someone else where we only have user rights. If this is the case and we do not have access to the L0 servers and hypervisor, we will be limited to operations on the L1 and L2 layers. L2 operations means migration of the nested guests. For most applications in production environments this is the desired functionality and is enough to justify the added complexity.

Nested support are in all cases a requirement, and although nested clouds work it is not all combinations that are fully functional at present. One defi-

nite requirement is that the L0 hypervisor supports nesting. This is the case in the most recent KVM platform. KVM uses hardware virtualization and can host another KVM hypervisor inside it, meaning both layers using KVM and hardware virtualization. But this is the only combination to give the second layer full hardware acceleration. All other possibilities for nested virtualization to function must use a paravirtual or dynamic binary translation second layer. If nested virtualization is not supported on the L0 layer, as for example in Amazon EC2, we are limited to a paravirtual L1 hypervisor. This means in the case of EC2 that we are limited to Xen for our L1 cloud nodes through the Xen-Blanket project. There are at the present no major public cloud providers that support nested hardware virtualization, but this might change when the technology matures and becomes standard. It might also be desirable for the public cloud providers to not implement nesting into their clouds to prevent more complex setups and for competitive reasons. Inter-cloud migration could be seen as a potential problem for the providers as it becomes easier to choose the cheapest platform. More flexibility can prevent vendor lock-in, increase competition and potentially reduce revenue for the provider.

5.2.2 Live migration

The nested cloud will give us the ability to live migrate across platforms. The second layer private cloud can span multiple different underlying platforms and separate the guest VMs from the physical L0 layer. A potential scenario could be that we own a small data center hosting an OpenStack cloud and using the KVM hypervisor. We then build a new, virtual, cloud on top of this physical cloud. The second cloud is managed by OpenNebula and uses Xen as its hypervisor. This two layered cloud is now under our full control. We can then live migrate one of the original, physical, cloud VMs to a different physical server. But this VM is now also a cloud node for our virtual cloud and all its guest VMs will also follow over to the new physical server. This is possible since we control all aspects of the setup and the bottom hypervisors are the same. But now we can imagine that we want to migrate our guest VMs to a different cloud provider outside our control. This decision could be motivated by a number of reasons, for example lack of capacity in the existing cloud or proximity to the customers using our services. It could also be for political or security and privacy reasons, or price in form of capacity rental or power consumption. Whatever the reason we do not have control over this foreign cloud and can not migrate the L1 nodes. We can also think that this other cloud uses Xen as its L0 hypervisor, which is not compatible with our KVM setup. The solution we have shown in this paper is to start virtual machines in the foreign cloud, prepare them with a provisioning tool, install it as a cloud node identical to our L1 cloud nodes and join them to our private nested cloud running OpenNebula. This enabled us to effortlessly live migrate the nested cloud guests to the foreign cloud. OpenNebula is not aware of that the underlaying platform has changed as the L1 layer hypervisor and environment stays the same.

As discussed we could not migrate the L1 cloud nodes to the foreign cloud as we did not have control over it with a shared storage and the L0 hypervisors was not the same. But we see promising projects[36] and initiatives in the field of live migration in heterogeneous environments. This will enable live migration of virtual machines between different hypervisors without shared storage. It is achieved by copying the raw disk-image and boot the VM up again with identical properties as the original VM. It enables close to live migration of VMs from one hypervisor to a different one without shared storage. The technique used to shorten the time of disk copying is called *dirty-blocks* or *dirty-files*. It is based on the similar method of migrating the memory to achieve live migration. The implications of such a successful system in this thesis' topic will mean that we can live migrate complete virtual data center across all major hypervisors with virtually no downtime. We can run our own controlled private cloud on top of any platform or existing cloud and span the cloud across them. Live migration of the nested guest VMs will be possible between all the underlying platforms, and with the addition of the *dirty-blocks* technique we will be able to migrate a complete virtual data center from one cloud to another without any common underlying platform.

5.3 Concept Application

Nested virtualization and nested clouds opens up a wealth of possibilities. It enables applications to be hosted in a private cloud in our control even though it runs on a underlying platform not in our control. This decoupling of the guest virtual machines and the physical hardware introduces layering and modularity to the cloud stack. The VMs are no longer tied to the original hypervisor and platform and layers can therefore be changed without redesigning the whole implementation. The application running on top of the cloud and hosting the service provided to the customers is often implemented into the cloud platform of choice. When this first deployment in production has been carried out it can be time consuming and costly to change cloud provider. Another hold-back is the increasingly complex operation of these clouds with different functionality and ways of solving issues. Other considerations when changing cloud providers are all the different laws, rights and agreements. As a result changing the cloud to host your services is a cumbersome and ineffective process contributing to the lock-in effect.

With nested clouds the application and services only interact with one private cloud, whatever the underlying platform. This cloud is chosen, designed and fully controlled by the user and can last for much longer than a changing third party cloud. The private cloud will be hosted inside other clouds or virtual environments which can be changed and migrated between. The process of changing the underlying platform will now be easier since it is not directly affecting the services hosted in the top virtual machines. And also because we now have full control over the private cloud we can better control data traffic and storage to ease the issues regarding laws and regulations.

This truly hybrid cloud with inter-cloud migration is one potential use-case for nested clouds. Another application of this nesting is in management of large data centers and private clouds. If we consider a large set of servers within our control, a privately owned data center, with a heterogeneous combination of hardware and software. Some servers can run Xen and OpenNebula while others uses the KVM hypervisor managed by the OpenStack cloud platform. And maybe an old part with VMware servers is not managed in a cloud at all, just a normal virtual environment. We can now create a private cloud on top of the whole center. This includes all types of hypervisor and cloud platform. From a single point we can now administrate the entire data center and live migrate virtual machines between all parts.

5.4 Impact in production

While virtualization has been widely adopted in the industry for some years, the use of clouds to organize the virtual environments is a fairly new approach. While clouds provide many organizational advantages and in the case of public clouds also provides on-demand financial benefits, clouds have certain challenges. These are primarily of a security and privacy nature, but also includes the risk of lock-in and dependence of specific function.

5.4.1 Security

The security will not automatically be increased with an additional virtual layer, even though the system becomes more complex and obscure. In a way nesting a hypervisor could contribute to higher security through sandboxing and isolation of the hypervisor. As the second hypervisor now is virtual it can more easily be isolated and changed in the case of a rootkit injection or other malicious code. It will not be possible to gain access to the L0 layer if the L2 layer is compromised, as it must pass through the virtual barricade created by the nested hypervisor. Likewise, if the L0 is breached, the L2 logic is safe since they are different systems. This will be beneficial if for example the internal isolation between users instances in Amazon EC2 is broken through and an instance is compromised. Normally such a breach will leave the application logic exposed to the attacker, but with a nested cloud the application is hidden behind an additional virtual layer. The same applies to a physical break-in and unauthorized personnel gaining access to the physical servers and L0. A further security benefit is the increased control the nested cloud will give the implementor. This applies especially to nested private clouds hosted in public clouds like EC2. Amazon has implemented a range of security features and provide tools and services to increase the security, but in general the users relay on trust and has confidence in Amazon's ability to maintain the security level they claim. Besides basic operating system protection like password, encryption and intrusion detection systems the user has little control over the security of their virtual instances. With nested cloud computing the second

hypervisor running inside the normal instance gives the user back full control over the private cloud stack. The nested guest (L2) VMs will be out of reach and control for Amazon. The increased control also means the user has more abilities to design and implement protection best suited to the needs. This can include encryption of the whole nested cloud and its communication with the other nodes, resulting in better protection if the L0 level where to be compromised. All the traffic to and from the nested guests could also be encrypted to obstruct eavesdropping from third parties.

5.4.2 Privacy

Privacy is also a major concern when moving business critical systems onto a cloud platform. The user will accept and sign a service level agreement and must comply with the applicable laws and legislations. These laws will vary depending on the geographical location of the cloud providers data center, and therefore also have different consequences for the user. This could permit local law enforcement to hold data hostage under criminal investigations or confiscate information for examination. By using nested cloud computing the user will regain the control of both the guest virtual machines and the cloud nodes and therefore be able to easier control the data flow and data storage. It could be implemented in a way that no data is stored in the public cloud, but fetched from a secure and trusted source. The traffic could be encrypted to and from the virtual machines. This way only the processing is taking place in the cloud, and in a case where the cloud provider seizes the users virtual machines there are no data stored on them so no data is lost or compromised. The user can simply start new virtual machines or start virtual machines on a different public cloud and maintain a functional service.

5.4.3 Vendor Lock-in

All platforms like clouds have the potential to lock the user to the specific provider. This is known as vendor lock-in and is highly relevant for clouds, and public clouds in particular. Lock-ins results in difficulty of changing provider. This is a result of different APIs and standards for running the virtual machines and the technology used in the data centers. As we have shown is it generally not possible to move a VM across hypervisor boarders. And as the hosting of the virtual machines are developed and administrated by the vendor there are no functionality to import or export virtual machines to a different provider. With nested clouds the used regain the administrative control and are free to implement a solution that allows moving VMs in or out of the public clouds. One such solution could be as described in this thesis: A second, nested, private cloud that spans across two or more public or private clouds. This way the public cloud lock-in is broken, and the user is free to choose and change provider. As public clouds have a growing offering of additional services like load balancing and monitoring the lock-in problem will grow. Usage of these extra features from the provider will not be easily migrated to a different cloud along with the virtual machines, and the services

will not necessarily function on the nested cloud VMs. They must therefore be implemented on the nested private cloud. The implementation of for example a load balancer on the nested level will be possible to migrate to the other cloud since it is running inside the user-owned private cloud.

5.4.4 Business Flexibility

With a fully flexible cloud environment virtual machines could live migrate across platforms. A thinkable scenario is a pool of different public and private clouds the user has access to as the base platform. With a provisioning tool, like Cfengine, Puppet, Juju, etc, virtual instances could be prepared on all the various clouds as cloud nodes for the virtual private cloud. When needed the virtual nodes would be joined into the nested cloud and be ready to host nested virtual guests. These virtual guests will then be able to live migrate between all the virtual cloud nodes and across the underlying private or public cloud platforms.

From a business perspective this would be advantages to ensure that the virtual machines, and thereby the service, are running on the most cost effective platform. And in the case of a price change the service could be live migrated to a different underlying platform to maintain the lowest possible cost.

When it is not desirable, or the service is not suitable, to be hosted in a public cloud the virtual machines could be limited to migrate between private clouds. The policy might then change from cost effectiveness to power efficacy. The datacenter with the lowest power consumption or with the cheapest electricity supply could be prioritized.

Equally could political unrest or company policy dictate the best suited host for the service. The virtual machines and applications will then be migrated based on these requirements while maintaining operation and without any reimplementation or porting.

The migration of virtual machines between platforms could be automated to best fulfill the desired policy. Regardless of the policy, the factors, like pricing, power, politics, etc, will be input to the automation logic which will make a decisions and migrate the virtual machines.

The system is also able to maintain high availability by ensuring that virtual machines are running on multiple underlying platforms at any given time. If one platform fails, the remaining platforms will continue to function. The instances on the failing system will either be migrated over to a different one, or new VMs are started on the healthy platform.

5.5 Future possibilities and work

For a nested cloud to be implemented in a production ready environment there are some more research to be done. This thesis has been concentrating on compute clouds, i.e. the hosting of virtual machines and live migration of these. For this to be sustainable, scalable and high performing more research must go into storage and networking. When starting a VM in the cloud it will boot from an image, and this image is stored in the shared storage reachable by all cloud nodes. But if some virtual machines are live migrated off-site and over great geographical distances the shared storage will be considerably slower and difficult to reach. A proposal is to partially migrate the storage as well with the VMs to maintain proximity to the storage. This could be achieved by using the previously mentioned *dirty-blocks*[36] method. There is however not always high traffic between the VMs and the image store, since the service running on the VMs is only working in memory and towards a database or similar. Networking will also be a challenge as the remote cloud could be incompatible with the subnets used for the VMs, resulting in unreachable machines. This could be solved in a number of ways, like dynamic DNS, tunneling or with an implementation of OpenFlow[8][24] and Open vSwitch[30][23]. If the public IPs are routable in the remote cloud the VMs will continue to work. There could also be implemented a solution of changing subnets when migrating across network boarders. OpenFlow could also potentially solve, or contribute to, the network challenges.

This thesis has also concentrated on hardware virtualization as the hypervisor for the performance tests and implementation. But since this solution not yet is supported in the public clouds it is necessary to use paravirtualization through Xen and the Xen-Blanket[38][37] project. This concept also needs more thorough testing to prove its usability without a significant loss of performance.

Combinations of live migration to and from public and private clouds also requires further testing. The performance levels must be established beyond the general function and feasibility covered in this thesis. Also large scale real scenario tests should be carried out to ensure the performance beyond the synthetic benchmarks in this paper. This could be an implementation of a private nested cloud running on top of a combination of several private and public clouds with a load balanced web application. This way it can be stress tested with real life traffic and the performance monitored for the nested VMs running on the different underlying platforms. Then parts of the nested cloud could be live migrated under load to a different platform to observe the behavior and measure the potential loss in performance.

Chapter 6

Conclusion

This thesis explored the possibility and potential of nested clouds. By using the recently added support for nested virtualization in hardware virtualization for the x86 platform, the hypervisor can host a second hypervisor on top with additional virtual machines. The purpose was to address the current lock-in situation seen when adopting private or public clouds to host services and business critical applications. The goal was also to increase the flexibility with inter-cloud migration by decoupling the hypervisor from the physical hardware. For this to be feasible the performance overhead of nested clouds was critical to justify the added flexibility.

The overall function of private nested clouds are very promising, as nested virtualization is supported by both AMD and Intel. By running a private OpenNebula cloud inside another private OpenNebula cloud, both using hardware virtualization, the tests shows convincing and consistent results for compute performance. The total overhead is about 5% higher compared with a traditional, one layered, virtualization. This is an acceptable loss of performance for nested clouds to be applicable in most cases, but is dependent of the systems performance requirements.

Nested private clouds enables greater flexibility in choice of the bottom layer hypervisor. As long as the layer 0 hypervisor has nested support the virtual cloud could live migrate virtual machines across heterogeneous platforms. The lack of support of nesting in the lower hypervisor preclude hardware virtualization on the second layer, this limits the nested cloud to paravirtualization. Using the Xen hypervisor in the nested cloud enables live migration of virtual machines to any underlying platform.

Using a nested cloud with Xen as hypervisor opens up for two layered clouds with a public cloud as the base. This thesis has shown that Xen virtual machines running in a private OpenNebula cloud can be live migrated to the Amazon EC2 public cloud. There are no technical limitations for the bottom layer, so any private or public cloud could host a virtual nested cloud inside using paravirtualization.

In the case of equal bottom layer hypervisors the thesis also shows that the entire virtual cloud, including all virtual machines, can be migrated to a new physical location.

A private cloud spanning different virtual environments, like clouds, could be called a true hybrid cloud. Features such as inter-cloud migration is now possible, and could facilitate cost savings as a result of easy switching of provider, repealing of vendor lock-ins and access to new platforms, as the service no longer interact directly with the underlying platform but with the private nested cloud.

Bibliography

[1] Hussam Abu-Libdeh, Lonnie Princehouse, and Hakim Weatherspoon. *RACS: A Case for Cloud Storage Diversity*. Cornell University Ithaca, NY, 2010.

[2] Cloud Security Alliance. *Top Threats to Cloud Computing*. Cloud Security Alliance, 2010.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. *A view of cloud computing*. Communications of the ACM 53.4, 2010.

[4] Esma Aïmeur, Sébastien Gambs, and Ai Ho. Upp: User privacy policy for social networking sites, 2009.

[5] Muli Ben-Yehuda, Michael D. Day, Zvi Dubitzky, Michael Factor, Nadav Har'El, Abel Gordon, Anthony Liguori, Orit Wasserman, and Ben-Ami Yassour. *The Turtles Project: Design and Implementation of Nested Virtualization*. Usenix OSDI 10, 2010.

[6] Lennart Berggen, Jonathan Borwein, and Peter Borwein. *Pi, a source book*. Springer-Verlag New York, 2003.

[7] Olivier Berghmans. *Nesting Virtual Machines in Virtualization Test Frameworks*. University of Antwerp, 2010.

[8] B Boughzala, R Ben Ali, M Lemay, Y Lemieux, and O Cherkaoui. *OpenFlow supporting inter-domain virtual machine migration*. Wireless and Optical Communications Networks (WOCN), 2011.

[9] Gerry Carr. *Ubuntu Server Survey*. Canonical, 2012.

[10] Jon Crowcroft. *Net Neutrality: The Technical Side of the Debate*. University of Cambridge, 2007.

[11] Andy Edmonds, Thijs Metsch, Alexander Papaspyrou, and Alexis Richardson. *Towards an Open Cloud Standard*. XX, 2012.

[12] EMC. *White Paper: Virtualizing Business-Critical Applications*. EMC, 2010.

[13] Rui Máximo Esteves and Chunming Rong. *Social Impact of Privacy in Cloud Computing*. Department of Electrical and Computer Engineering University of Stavanger, 2010.

[14] Bhagyaraj Gowrigolla, Sathyalakshmi Sivaji, and M.Roberts Masillamani. *Design and Auditing of Cloud Computing Security*. Hindustan Institute of Technology and Science, 2010.

[15] Zhiyun Guo and Meina Song. *A Governance Model for Cloud Computing*. School of Computer, Beijing University of Posts and Telecommunications, 2010.

[16] Apache Software Foundation / Red Hat. deltacloud.apache.org.

[17] Wassim Itani, Ayman Kayssi, and Ali Chehab. *Privacy as a Service: Privacy-Aware Data Storage and Processing in Cloud Computing Architectures*. Department of Electrical and Computer Engineering American University of Beirut, 2009.

[18] W. Pitt Turner IV and Kenneth G. Brill. *Cost Model: Dollars per kW plus Dollars per Square Foot of Computer Floor*. Uptime Institute, 2008.

[19] Ali Khajeh-Hosseini, Ian Sommerville, and Ilango Sriram. *Research Challenges for Enterprise Cloud Computing*. Cloud Computing Co-laboratory School of Computer Science University of St Andrews, 2010.

[20] B. Khasnabish and C. JunSheng. *Cloud/DataCenter SDO Activities Survey and Analysis*. ZTE USA, 2011.

[21] Michael Larabel. *Ubuntu 12.04 KVM/Xen Virtualization: Intel vs. AMD*. phoronix.com, 2012.

[22] H. C. Lauer and D. Wyeth. *A recursive virtual machine architecture*. Proceedings of the workshop on virtual computer systems, 1973.

[23] Yan Luo. *Network I/O Virtualization for Cloud Computing*. IT Professional, 2010.

[24] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. *OpenFlow: enabling innovation in campus networks*. ACM SIGCOMM Computer Communication Review 38.2, 2008.

[25] Chandrakant D. Patel and Amip J. Shah. *Cost Model for Planning, Development and Operation of a Data Center*. HP Laboratories Palo Alto, 2005.

[26] Siani Pearson. *Taking Account of Privacy when Designing Cloud Computing Services*. HP Laboratories, 2009.

[27] Siani Pearson and Azzedine Benameur. *Privacy, Security and Trust Issues Arising from Cloud Computing*. HP Laboratories, 2010.

[28] Lucian Popa, Sylvia Ratnasamy, Gianluca Iannaccone, Arvind Krishnamurthy, and Ion Stoica. *A Cost Comparison of Data Center Network Architectures*. University of California, Berkeley, 2010.

[29] Parikshit Prasad, Badrinath Ojha, Abhishek Vaish, Rajeev Ranjan shahi, Utkarsh Goel, and Ratan Lal. *3 Dimensional Security in Cloud Computing*. Indian Institute of Information Technology, 2011.

[30] Yan Pu, Yilong Deng, and A Nakao. *Cloud Rack: Enhanced virtual topology migration approach with Open vSwitch*. Information Networking (ICOIN), 2011 International Conference, 2011.

[31] Forrester Research. *Virtual Security In The Data Center*. Forrester Research, 2012.

[32] Francisco Emanuel Liberal Rocha. *Privacy in Cloud Computing*. UNIVERSIDADE DE LISBOA, 2010.

[33] Konstantinos K. Stylianou. *An Evolutionary Study of Cloud Computing Services Privacy Terms*. John Marshall Journal of Computer and Information Law, 2010.

[34] Symantec. *Adoption of Server Virtualization Widespread: Survey*. Symantec, 2011.

[35] Job Timmermans, Veikko Ikonen, Bernd Carsten Stahl, and Engin Bozdag. *The Ethics of Cloud Computing*. Technical University of Delft, 2010.

[36] Eirik Vada. *Creating Flexible Heterogeneous Cloud Environments*. University of Oslo, 2012.

[37] Dan Williams, Eslam Elnikety, Mohamed Eldehiry, Hani Jamjoom, Hai Huang, and Hakim Weatherspoon. *Unshackle the Cloud*. IBM T. J. Watson Research Center, Hawthorne, NY, 2011.

[38] Dan Williams, Hani Jamjoom, and Hakim Weatherspoon. *The Xen-Blanket: Virtualize Once, Run Everywhere*. IBM T. J. Watson Research Center, Hawthorne, NY, 2012.

[39] Minqi Zhou, Rong Zhang, Wei Xie, Weining Qian, and Aoying Zhou. *Security and Privacy in Cloud Computing*. Software Engineering Institute, East China Normal University Shanghai 200062, China, 2010.

Appendix A

libvirt VM config

Cloud Node on AMD (w/virtio)

```
<domain type='kvm'>
  <name>large</name>
  <uuid>5b2d6cb2-25f3-6b21-6091-158ec93b3263</uuid>
  <memory>30720000</memory>
  <currentMemory>30720000</currentMemory>
  <vcpu>7</vcpu>
  <os>
    <type arch='x86_64' machine='pc-0.14'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <cpu match='exact'>
    <model>Opteron_G3</model>
    <vendor>AMD</vendor>
    <topology sockets='1' cores='7' threads='1'/>
    <feature policy='require' name='wdt'/>
    <feature policy='require' name='skinit'/>
    <feature policy='require' name='osvw'/>
    <feature policy='require' name='3dnowprefetch'/>
    <feature policy='require' name='cr8legacy'/>
    <feature policy='require' name='extapic'/>
    <feature policy='require' name='cmp_legacy'/>
    <feature policy='require' name='3dnow'/>
    <feature policy='require' name='3dnowext'/>
    <feature policy='require' name='pdpe1gb'/>
    <feature policy='require' name='fxsr_opt'/>
    <feature policy='require' name='mmxext'/>
    <feature policy='require' name='ht'/>
    <feature policy='require' name='vme'/>
```

```
</cpu>
<clock offset='utc'/>
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<devices>
 <emulator>/usr/bin/kvm</emulator>
 <disk type='file' device='disk'>
  <driver name='qemu' type='raw'/>
  <source file='/var/lib/libvirt/images/large.img'/>
  <target dev='hda' bus='virtio'/>
  <address type='pci' domain='0x0000' bus='0x00' \
                      slot='0x05' function='0x0'/>
 </disk>
 <disk type='block' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <target dev='hdc' bus='ide'/>
  <readonly/>
  <address type='drive' controller='0' bus='1' unit='0'/>
 </disk>
 <controller type='ide' index='0'>
  <address type='pci' domain='0x0000' bus='0x00' \
                      slot='0x01' function='0x1'/>
 </controller>
 <interface type='bridge'>
  <mac address='52:54:00:84:11:02'/>
  <source bridge='br0'/>
  <model type='virtio'/>
  <address type='pci' domain='0x0000' bus='0x00' \
                      slot='0x03' function='0x0'/>
 </interface>
 <serial type='pty'>
  <target port='0'/>
 </serial>
 <console type='pty'>
  <target type='serial' port='0'/>
 </console>
 <input type='mouse' bus='ps2'/>
 <graphics type='vnc' port='-1' autoport='yes'/>
 <video>
  <model type='cirrus' vram='9216' heads='1'/>
  <address type='pci' domain='0x0000' bus='0x00' \
                      slot='0x02' function='0x0'/>
 </video>
 <memballoon model='virtio'>
  <address type='pci' domain='0x0000' bus='0x00' \
                      slot='0x04' function='0x0'/>
```

```
81      </memballoon>
82    </devices>
83  </domain>
```

Cloud Node on Intel (w/virtio)

```
<domain type='kvm'>
  <name>nested</name>
  <uuid>196c8fa8-9180-319c-240c-85852a29d043</uuid>
  <memory>1048576</memory>
  <currentMemory>1048576</currentMemory>
  <vcpu>2</vcpu>
  <os>
    <type arch='x86_64' machine='pc-1.0'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
    <apic/>
    <pae/>
  </features>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/bin/kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw'/>
      <source file='/var/lib/libvirt/images/nested.img'/>
      <target dev='hda' bus='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' \
                                slot='0x05' function='0x0'/>
    </disk>
    <disk type='block' device='cdrom'>
      <driver name='qemu' type='raw'/>
      <target dev='hdc' bus='ide'/>
      <readonly/>
      <address type='drive' controller='0' bus='1' unit='0'/>
    </disk>
    <controller type='ide' index='0'>
      <address type='pci' domain='0x0000' bus='0x00' \
                                slot='0x01' function='0x1'/>
    </controller>
    <interface type='bridge'>
      <mac address='52:54:00:db:1e:08'/>
      <source bridge='br0'/>
      <model type='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' \
                                slot='0x03' function='0x0'/>
    </interface>
```

```
    <serial type='pty'>
      <target port='0'/>
    </serial>
    <console type='pty'>
      <target type='serial' port='0'/>
    </console>
    <input type='mouse' bus='ps2'/>
    <graphics type='vnc' port='-1' autoport='yes'/>
    <video>
      <model type='cirrus' vram='9216' heads='1'/>
      <address type='pci' domain='0x0000' bus='0x00' \
                          slot='0x02' function='0x0'/>
    </video>
    <memballoon model='virtio'>
      <address type='pci' domain='0x0000' bus='0x00' \
                          slot='0x04' function='0x0'/>
    </memballoon>
  </devices>
</domain>
```

Nested Guest (w/virtio)

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas\
                    /domain/qemu/1.0'>
  <name>one-82</name>
  <uuid>56706685-003c-2c4b-9cf0-ea884139d604</uuid>
  <memory>524288</memory>
  <currentMemory>524288</currentMemory>
  <vcpu>1</vcpu>
  <os>
    <type arch='x86_64' machine='pc-0.14'>hvm</type>
    <boot dev='hd'/>
  </os>
  <features>
    <acpi/>
  </features>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>destroy</on_crash>
  <devices>
    <emulator>/usr/bin/kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='raw'/>
      <source file='/var/lib/one//82/images/disk.0'/>
      <target dev='hda' bus='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' \
                                slot='0x05' function='0x0'/>
    </disk>
    <controller type='ide' index='0'>
      <address type='pci' domain='0x0000' bus='0x00' \
                                slot='0x01' function='0x1'/>
    </controller>
    <interface type='network'>
      <mac address='02:00:0a:00:01:b4'/>
      <source network='default'/>
      <model type='virtio'/>
      <address type='pci' domain='0x0000' bus='0x00' \
                                slot='0x03' function='0x0'/>
    </interface>
    <input type='mouse' bus='ps2'/>
    <graphics type='vnc' port='5982' autoport='no' listen='0.0.0.0'/>
    <video>
      <model type='cirrus' vram='9216' heads='1'/>
      <address type='pci' domain='0x0000' bus='0x00' \
                                slot='0x02' function='0x0'/>
    </video>
```

```
46      <memballoon model='virtio'>
47        <address type='pci' domain='0x0000' bus='0x00' \
48                                slot='0x04' function='0x0'/>
49      </memballoon>
50    </devices>
51  </domain>
```

Appendix B

Benchmarking

pi.pl CPU Benchmark script

```
#!/usr/bin/env perl
# Using "Gregory-Leibniz":
# http://en.wikipedia.org/wiki/Leibniz_formula_for_pi

use Time::HiRes qw(time);
use List::Util qw(sum);

# Intercepting Ctrl-C singnal SIGINT and calling the quit sub
$SIG{INT} = \&quit;


open(FILE,">pi.log");

my $i = 0;
for(1..200) {
	my $t = time;

	$numerator = 4;
	$denominator = 1;
	$negVal = 1;
	$pi = 0;

	# Number of iterations for loop to calculate pi
	$num = 10000000;

	for (1 .. $num) {
		$pi += $negVal * ($numerator/$denominator);
		$negVal *= -1;
		$denominator +=2;
	}


	my $elapsed = time - $t;
```

```
34      print "$i $elapsed\n";
35      print FILE "$i $elapsed\n";
36      $i++;
37      $avg[$i] = $elapsed;
38  }
39
40  quit();
41
42  sub quit {
43      print "Sum: ".sum(@avg)/(scalar(@avg)-1)."\n";
44      close(FILE);
45      exit(0);
46  }
```