UNIVERSITY OF OSLO
Department of Informatics

Automation and Abstraction for Scalable z/VM Linux Administration on the zSeries Mainframe

Master thesis

Marius B. Gundersen

May 2008

Automation and Abstraction for Scalable z/VM Linux Administration on the zSeries Mainframe

Marius B. Gundersen
Oslo University College

May 19, 2008

Abstract

This thesis considers the administration of virtual machines on IBM mainframes running z/VM. A solution for administrating z/VM through a Linux VM running on a custom designed z/VM architecture is developed and implemented. The administration tool used is a slightly expanded version of MLN. The expansions added allows MLN to utilize plugins for technology specific code. Support for z/VM are then added through the creation and introduction of a plugin containing all z/VM specific code. Results from scenarios conducted shows that the administration process can be significantly automated and abstracted from a normal z/VM perspective. Also, increased security and safety is achieved through the protective limitations and control offered by the Programmable Operator running on z/VM.

Acknowledgements

First and foremost I would like to thank my advisor Kyrre Begnum for his guidance through this project. I could hardly have asked for a better supervisor as his moral and technical support and insight has been of incalculable help and a motivational driving force in itself. I cannot express with words the gratitude I feel for his enthusiasm and all the time he set of for me. I am truly proud of being his advisee, and truly grateful for getting to know him as a person. Also, thanks to his lovely wife Miriam for allowing me to take up so much of his time during their wedding period and for the coffee.

I would also like to extend my thanks to all the lecturers on the Network and System Administration program for two amazing years. They have done an outstanding job at imparting their knowledge and preparing us students for this project. I would especially like to mention Professor Mark Burgess who through his course "Analytical Network and System Administration" and many informal talks have opened my eyes to what it truly means to be a scientist and the analytical mindset. It has been an enormous asset in this project and I am sure it will serve me well in the future.

I have also been graced with great classmates. They have been fine companions on this journey of academic enlightenment, and I treasure the time we have spent together both at the college and outside it.

I was lucky enough to do this project in cooperation with IBM, and for that I am extremely grateful. It would not have been possible to complete the project without their support, and I am proud to have worked with the company. I would like to emphasize three persons in particular (in no particular order).

IBM Certified Senior IT Specialist Per Fremstad has been my main contact person in IBM. I can safely say that I would not have done this project had it not been for him, the reason being quite simple. Before I met Per, I had no knowledge at all about mainframes. It was Per and his course "Supercomputers and Virtual Operating Systems" that introduced me to the incredible world of mainframes, and for that I will always be grateful. Per has been an amazing person to work with. He has never hesitated to set

aside time to talk with me and give me advice, and his bottomless well of enthusiasm for the mainframe has been an inspiration to me.

My first real encounter with z/VM was through a lecture held by Linux Technical Consultant Malcolm Beattie, but although that was an important event for me, it is not the main reason I speak of him here. The main reason this could be a practical project is thanks to Malcolm. It is on "his" mainframe environment that this entire project has taken place. Not only did he give me access to the mainframe, he also gave me free access to IBM proprietary software. He has also provided crucial practical assistance, guidance and knowledge during the project. For all of this, I cannot thank him enough.

I had the pleasure of getting to know IT Specialist Per Rosenquist during a z/VM course IBM let me attend. Being a z/VM expert in every sense of the word, he taught me a great deal about the technology. Most importantly, it was not until after listening to Per that I felt truly comfortable and familiar in the new z/VM environment. It did not stop there however, as Per continued to be immensely helpful by always answering every mail I sent him quickly and thoroughly. Many a problem were solved with the help of his expert advice, and for that he has my utmost gratitude.

May, 2008

Marius B. Gundersen

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The topic of this project is administrating Linux virtual machines[1] on IBM mainframes (mainly zSeries). Mainframes have received a lot of attention lately when it comes to consolidating servers to a single system. There are multiple benefits from this, from environmental and saving space to the ability to utilize hardware to a much greater extent. For a system administrator who is mainly used to multiple Intel Unix servers however, migrating to a mainframe platform is inherently different.

- New operating system. The zSeries mainframes primary operating system is z/OS, an OS custom created for the mainframe hardware.
- New virtualization technology. For virtualization, zSeries mainframes run z/VM. As the operating system, it is custom created for the mainframe.
- New interface. z/OS can be controlled either through command line or ASCII-like GUI screens. z/VM is controlled exclusively through command line.
- New filesystem. Both z/OS and z/VM uses a record based filesystem, not byte based like Unix or MS Windows.

It should be apparent from these points that a mainframe is more than just a really big PC.

z/OS and z/VM might seem difficult and overly complicated to use and administrate for someone without mainframe experience. Even for experienced system programmers it can be quite time-consuming to manage a large number of VMs. What can be done about this problem?

[1]The term "guest" is commonly used when referring to virtual machines on z/VM. The terms "virtual machine", "VM" and "guest" will be used interchangeably throughout this text.

The purpose of this project is to design and create a z/VM environment where an expanded version of MLN can be implemented in a designated administrator Linux guest. Thereby giving system administrators without z/VM experience a more familiar environment to work with, and automate tasks for system programmers. The motivation is that mainframes are the ideal machines for the purpose of running a large amount of VMs (the z mainframes have been tested with tens of thousands of VMs running at the same time), and most of these VMs will be identical clones. Creating and administrating multiple identical VMs is exactly what MLN was designed to do. Therefore it seems logical to combine these technologies.

1.1.1 Personal reasons for wanting to do this project

- I would very much like to do a project involving mainframes, but it would be nice if it didn't mean completely losing touch with the Unix platform. This project involves both IBM mainframes and Linux, the best of both worlds so to speak.
- I have for some time now wanted to learn Perl as I believe it is a great asset to any system administrator. This project would give me a much desired push to really learn Perl instead of just playing around with it.
- As mentioned before, I would like a project involving mainframes. This project gives me a chance to not only refresh some z/OS knowledge, but also to learn more about z/OS and z/VM.
- Virtualization has received a lot of attention the last several years, and justly so. The possibilities with this branch of technology are staggering and I would definitely like to learn more about it.

1.2 Problem definition

1. *Can the management of a high number of Linux VMs under z/VM be simplified through increased automation?*
   - *I) Can MLN be modified/expanded to offer its original functionality for z/VM?*
   - *II) Can a suitable z/VM environment architecture be implemented for use by MLN?*
   - *III) If the first two can be achieved, can the benefits be demonstrated through general scenarios?*

Some terms in the problem statement deserve concrete definitions:

Management. Tasks which are performed relatively often in an environment dependent on VMs. Examples would be starting, stopping, creating and destroying VMs as well as connecting the VMs in virtual networks.

High number. On a mainframe the number of VMs can easily be in the range of several hundred.

Simplified. This will be measured by a number of metrics. Most notably the number of commands, lines and characters. The complexity of the tasks and commands will also be considered.

Scenarios. Fictional situations in which the different approaches are considered.

Beneficial. The benefits will mostly be decided by the reduction of workload for the administrator.

Increased automation. By increased automation steps will be grouped into tasks, thereby reducing the number of operations the user initiates. This will be compared to the level of automation determined before the proposed solution is implemented.

Modify/expand. Writing necessary plugins for MLN and detect possibly necessary architectural expansions to original MLN codebase.

Original Functionality. Functions already implemented in MLN at project start.

Compared to. Each parameter related to the simplification and automation will be considered by itself. Further, the number of steps and commands, which are related to the workload of a system administrator, will be of particular interest.

1.3 Approach

The project will be divided into four distinct parts

1. **Design**
   Before the practical development of the system can begin, the system must first be designed. Different approaches must be considered and decisions about the underlying z/VM platform must be made. Whether the expansion of MLN occurs through the creation of a plug-in or a completely integrated module must also be decided.

2. **Implement**
   During this phase MLN will be expanded to work with z/OS and z/VM. It is important here to consider the shift from i86 to z/360 architecture. Perl will be used as this is the original language of MLN. The underlying z/VM architecture will also be implemented at this point. This is the phase most prone to potential problems. For this part to be possible and realistically carried out, access to an IBM z

mainframe is crucial. The difference in architecture and file systems might also pose to large of an obstacle to overcome. Should this be the case, an important part of this project will be to document the attempt and investigate possible solutions on a lower lever of the system to lay the foundation for the stated primary objectives.

3. **Measure**
   After the completion of phase two, the product will be used in a measured comparison.

   - Compare the administration process of creating and administrating a large number of identical z/VM servers with and without MLN.

4. **Analyze/discussion**
   When phase three is complete, the metrics must be analyzed and discussed to derive meaning and conclusions.

1.4 Summary of results

A z/VM architecture was successfully developed supporting automation from a Linux VM. A plugin for MLN was created to take advantage of the z/VM layer and MLN itself was modified to allow the use of plugins for technology specific code. Three scenarios were carried out, the results indicating that the majority of the measured areas has been simplified and automated by the new system.

1.5 Thesis outline

This thesis considers the administration of virtual machines on IBM mainframes running z/VM. An introduction the technology is given in Chapter 2. The decisions and design of the solution, most notably the architecture of the z/VM layer, is described in Chapter 3. In Chapter 4 the actual implementation of the system is described. Scenarios are also conducted to estimate the change the system has had on the administration aspect. The system and project are then discussed in Chapter 5 before the conclusion is presented in Chapter 6.

Chapter 2

Background: Virtualization, MLN and Mainframes

In this chapter, an introduction will be given to virtualization in section 2.1 before moving on to virtual machine management in section 2.2. An introduction to virtualization on the mainframe is then given in section 2.3 and a few IBM projects are introduced in section 2.4. I the final sections, 2.5 and 2.6, MLN is introduced and some reasons for using MLN on the mainframe is mentioned.

2.1 Virtualization

For those not familiar with virtualization in computer science, a short introduction is in order. By virtualization one refers to the abstraction of the hardware layer, with the desired effect of simulating a different physical architecture. There are several benefits with doing this, and several different ways of virtualization.

2.1.1 Types of abstraction

Virtualization allows for several types of abstractions, three of which will be presented here.

One to many
Here we start with one physical resource. This resource can be everything from one mainframe to one hard disk. Through virtualization we abstract this entity to appear as several physical resources [Figure 2.1]. VMs in themselves are examples of this. By virtualizing a mainframe it is possible to have it appear as several physical machines. This form of abstraction can happen by dividing up the physical resources in parts, giving each virtual entity one part, or by time shar-

ing the physical resource among the virtual entities.

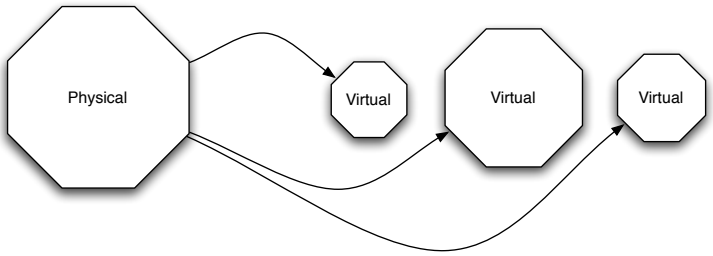

Figure 2.1: *One physical resource is abstracted into several virtual resources.*

Many to one

In this case the intention is to make several physical resources (of the same type) appear as one [Figure 2.2]. A common example is the abstraction of multiple storage devices. By virtualizing these devices they get the appearance of one storage area. This can then be shared among multiple users/systems, giving the possibility of making shares with a higher capacity than any of the individual physical devises. A common example of this is hard drives connected in a RAID.

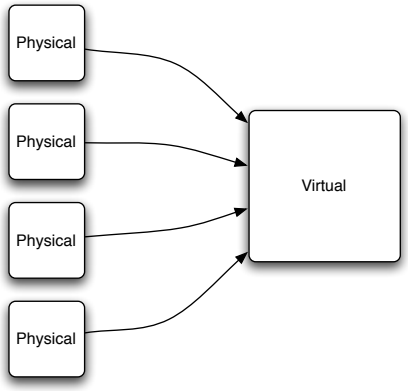

Figure 2.2: *Many physical resources are abstracted into one virtual resource.*

Something to something else

The purpose of this form of abstraction is not to change the apparent quantity of a resource, but rather to change the characteristics and attributes of a single device [Figure 2.3]. This is extremely useful to

make software compatible with hardware. If a software depends on a specific device to function or adheres to a certain hardware standard, the needed hardware can be virtualized to ensure compatibility. This is of great help when backward compatibility is an important issue (as is the case with IBMs mainframes).

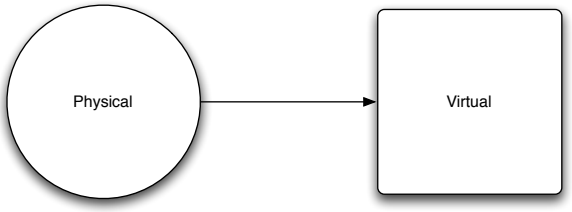

Figure 2.3: *One physical resource is abstracted into another virtual resource.*

2.1.2 The technology

Physical partitioning

The process of physical partitioning is simply the approach of dividing the hardware between partitions physically. As the name implies this happens on the physical level without the use of software or firmware. This method is becoming less and less common these days, and can not justifiably be viewed as virtualization. It is included in this text simply to give a more complete picture of partitioning.

Logical partitioning

Through logical partitioning, the hardware resources are divided by means of firmware. This allows for more freedom and flexibility compared to physical partitioning. Although still on the physical level, one can say that logical partitioning takes place on a higher aspect of the layer since it utilizes firmware. Whether hardware resources can be shared among logical partitions and reconfigured dynamically, as opposed to dedicated, statically allocated, depends on the logical partitioning technology used. Logical partitioning is often referred to as 'physical hypervisor'.

Software partitioning

In software partitioning, a hypervisor is used in the software layer to partition the system. This form of partitioning allows for a highly fine-grained and dynamic administration of resources. There are two different forms of hypervisors on this level:

Hypervisor Type 1
A Type 1 hypervisor have the hypervisor function implemented in the operating system itself. This makes the guest system run on the second level above the hardware. Xen [1] is an example of a Type 1 hypervisor.

Hypervisor Type 2
A Type 2 hypervisor runs as an application on top of the host operating system. As a result, the guest system runs on the third level above the hardware. User-Mode Linux [2] is an example of a Type 2 hypervisor.

2.2 Virtual machine management

Virtualization brings with it both advantages and disadvantages when it comes to operating and managing computer systems. It is important to know and understand both sides of the technology so as not to cause more damage than good when implementing it.

2.2.1 The benefits

When considering the benefits of virtualization, it is important to remember that they are context sensitive. To what degree a system will get these benefits, or if it will get some of them at all, is largely dependent on the architecture of the system.

Consolidation
Virtualization makes it possible to consolidate several physical machines on one single physical machine. The abstraction of the one machine allows it to function as several independent machines. This makes it possible to run several operating systems, identical or different, at the same time independently of each other. Thereby utilizing the hardware more effectively.

Lower management cost
Virtualization can lower management cost by reducing the amount of physical hardware that must be maintained and managed. It can also simplify management through abstraction and thereby decrease the time spent on management tasks.

Higher resource utilization
By pooling resources/consolidation, resources can be utilized much more dynamically. This is especially useful when it comes to resource requirements with a tendency to spike. It is not unusual for a normal server to utilize only about 20% of its assigned hardware resources.

The remaining 80% is only used to handle occasional spikes. This is a poor resource utilization since most of the time the 80% remains unused. Through virtualization, the necessary resources to handle spikes can be allocated to the server only when needed, while the rest of the time the resources are used elsewhere.

Increased flexibility
Through virtualization it is no longer necessarily a requirement for an administrator to wait for approval and purchase of hardware to set up an extra server or user machine. On a virtualization server machines can be created and destroyed in less time than it would take to set up a physical machine.

Finer resource allocation
Because of the virtualization of hardware, resources can be allocated with greater freedom. The size of memory modules and CPUs are no longer relevant as far as the VMs are concerned. If it is desirable to create a VM with 1701 MB of memory then that is just as simple as creating a VM with 256 or 512 MB of memory. Virtualization also incorporates the possibility to make changes to the resources during runtime.

Migration
If certain requirements (hardware consistency) are met within the system, it is possible to live migrate VMs. VMs have the ability to migrate between physical hardware in mainly two ways. In 'cold migration' the running VM is frozen and moved to a different location where it is unfrozen. In live migration the VM is moved with no downtime at all. In fact, a user on the VM will not notice that a migration has taken place. In addition to identical hardware on the two sites, live migration also require the filesystem to reside on a SAN[1].

2.2.2 Challenges with virtualization

Although beneficial in several cases, virtualization also introduces new problems and challenges. As with the benefits, it largely depends on the architecture of the setup as to what degree these challenges presents themselves.

Single point of failure
By consolidating several services/machines on one physical machine, it is easy to end up with a single point of failure. Should this physical machine fail, all the virtual systems contained on it naturally also fail.

[1]It should be noted that solutions where the filesystem resides completely on a RAM drive are not necessarily dependent on a SAN.

Overhead
Although there are virtualization platforms that provides a very high utilization of the hardware for the virtual environment, it is difficult to achieve the same utilization as a non-virtual, straight-on-the-hardware system. This overhead is difficult to avoid because the virtualization platform is a layer between the hardware and the VMs, and a system in itself that require resources to run.

Resource bottlenecks
When running several VMs on the same hardware, it is important to remember the limitation of said hardware. This is especially important when it comes to hardware that scales badly. The benefit 'Higher resource utilization' can backfire if spikes occur on several VMs at the same time. Parts of the system, especially I/O, can then turn into bottlenecks, decreasing performance dramatically. (Interestingly, I/O is something the z mainframes were design to excel at)

Number of systems
Through virtualization, one extra system is effectively added to the total number of managed systems. It is therefor necessary to consider if virtualization will improve the current situation. After all, virtualizing only one machine effectively doubles the number of systems that must be managed and maintained.

Non-scaling management
Although virtualization makes it possible to create extremely large and complex environments, the management does not necessarily scale as well. Several aspects of management and maintenance are done manually, and are therefor unsuited to these environments. Because of this, the full potential of virtualization may be lost in the impossibility of administrating the final product.

OS restrictions
The different virtualization platforms are currently able to support a wide variety of guest operating systems. However, there are still restrictions one several of them. Some of them can for example only support specific unix based OSs. This is important to take into consideration when choosing a virtualization platform. A change in the choice of platform to support a specific OS can lead to huge costs in time and money if a different platform is already implemented.

As mentioned in the drawbacks of virtualization, virtual machine management is not necessarily an area that scales well. In its basic form, VM management consists of giving individual commands to individual VMs. There are of course systems which greatly improves the aspect of managing a high number of VMs, but these are mostly proprietary and not open for

customization by the user. As a result, administrators often end up writing crude, undocumented scripts in languages like Perl or Python to handle specific tasks. This is an undesired solution because it hinders consistency and standards in the virtualization community, and because it makes the specific administrator 'indispensable' in his position if the scripts are to be utilized. There are however some free, open source projects, like MLN, that try to improve the situation. These projects attempt to offer a free alternative to the proprietary systems, thereby giving anyone administrating VMs a consistent, quality tested tool to work with.

2.3 Virtualization on the mainframe

Although virtualization has received a lot of attention the last years, it is by no means a "new thing". The history of virtualization stretches back almost fifty years to the late fifties early sixties [3]. In 1959 the scientific community, largely on Massachusetts Institute of Technology (MIT), began to explore the possibility of time sharing large IBM machines. The result was the creation of the Compatible Time Sharing System (CTSS) which was demonstrated on a IBM 709 processor in 1961 [3]. This can be seen as the birth of the VM.

After the introduction of CTSS, IBM began to develop the hardware to compliment this new technology. Also, some problems that could not be solved on the software level were instead solved on the hardware level. Because nearly fifty years of continuous development is more than enough to fill entire volumes by itself, it will be omitted in this text for the sake of the reader. Sufficient to say, it has all culminated to todays version of IBMs virtualization platform, z/VM 5.3.

2.3.1 LPAR

LPARs, or Logical PARtitions, are virtual machines created at the hardware level as described under 'Logical partitioning' in the 'The technology' subchapter (2.1.2). The creation and running of LPARs are handled by the Processor Resources/System Manager (PR/SM) facility. From a practical viewpoint, LPARs can be seen as separate physical mainframes with their own hardware and independent operating systems. This is not to say that the partitioning of the actual physical hardware is necessarily static. Through PR/SM, the system administrator can choose to share certain resources like I/O devices and processors among several LPARs. Modern mainframes always have at least one LPAR, and the System z mainframe supports up to 60 LPARs. Because LPARs run independently of each other, they can be started, stopped and even crash without affecting the other LPARs on the system. This makes it common to have for instance one 'development', one

'test' and one 'production' LPAR.

2.3.2 z/VM

z/VM is IBMs software level virtualization platform. It is an operating system with hypervisor incorporated, making it a Type 1 hypervisor (z/VMs hypervisor is called Control Program, or CP). z/VM normally runs on top of an LPAR or on another z/VM instance. This makes the VMs themselves run on the second level or higher as seen from the hardware level. z/VM supports 64-bit IBM z/Architecture and 31-bit IBM Enterprise Systems Architecture/390 guests [3]. This makes it capable of hosting z/OS, z/VM and Linux distributions adapted to System z, to name a few. z/VM allows for dynamic changes to several of its VMs resource attributes. This makes it possible to reconfigure VMs without rebooting them, potentially disrupting the use and running of the virtual systems.

It should be noted that older mainframes has the option to run in a 'basic' mode. In this mode the mainframe has no LPAR and instead runs z/VM or z/OS directly on the hardware. By doing this the system naturally looses the benefits and features of running on an LPAR. Newer mainframes do not support this option.

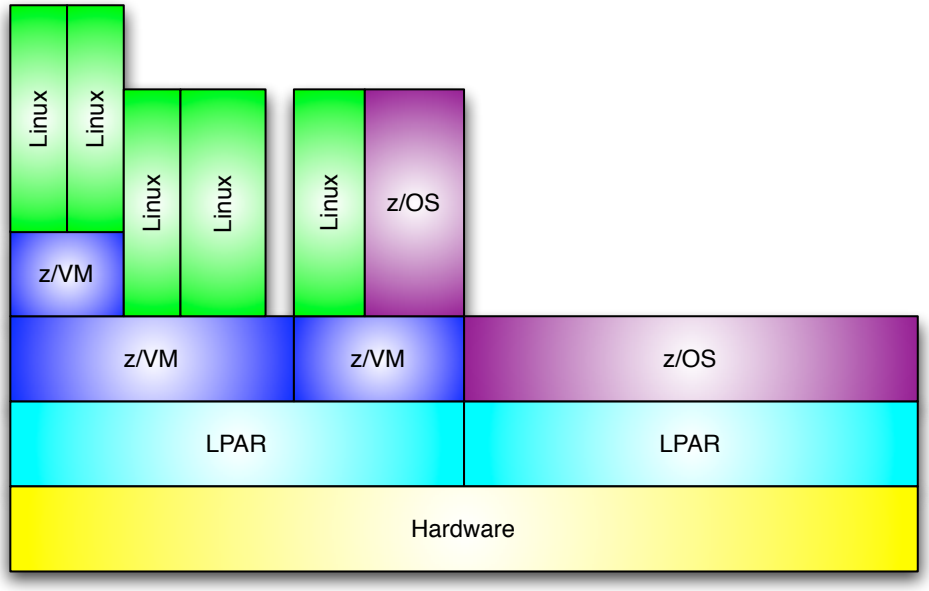

Figure 2.4: *Example of a virtualized environment on a Series z mainframe.*

2.3.3 Terms and Acronyms

There are some IBM z/VM specific terms and acronyms that will be used through this thesis. For the readers not already familiar with z/VM, a short description of the of these are in order. For those already familiar with the terms, feel free to skip this part as the acronyms should be consistent with the ones used in most IBM documentation.

Control Program

The Control Program will throughout this thesis be referred to by the acronym CP [3]. CP is one of a z/VM systems two primary components, the other being CMS which is covered in the next section. As mentioned before, CP is a Type 1 hypervisor and can itself be seen as an operating system. Note that in this context the term 'operating system' is not used to describe a software bundle like GNU/Linux or Microsoft Windows, it is more accurate to compare it to the Linux kernel by itself. The task of CP is to administrate the resources given to z/VM, it also handles the creation and administration of VMs. It is CP that takes the physical hardware and virtualizes it so that it can be used by the VMs. Although an operating system in itself, CP would be quite useless to a normal user. It can not be used to surf the internet or write text documents. In fact, CP alone does not even understand the concepts of files and processes. If CP is to run programs and services that are not part of CP, it does so through a dedicated service virtual machine.

CP is the greatest single point of failure in an z/VM environment. Failure in this low level can propagate through all the higher levels and destabilize the entire z/VM environment, even making it completely unavailable to the users. The greatest threat to CP can be seen as a careless (or new and unlucky) superuser. One careless command can easily take down the system. Even if it is just a simple shutdown, not permanently damaging the system, it will still have terminated all activity on all VMs on the system.

Conversational Monitor System

The Conversational Monitor System will throughout this thesis be referred to by the acronym CMS [3]. CMS is the second primary component of a z/VM system, the first being CP covered in the previous section. CMS is considered to be the default z/VM guest operating system and greatly increases the functionality of the system compared to a system only running CP. Where CP can be compared to the Linux kernel, CMS can be loosely compared to the Linux shell. It incorporates a high number of commands, and more closely resembles a basic operating system on a personal computer. It allows a user to create and edit files, execute applications and share data with other guest operating systems. CMS has a close relation-

ship with CP that allows a user to issue commands from CMS and directly to CP, this is useful and necessary when administrating VMs under z/VM.

In one aspect CMS can also be compared to GRUB (Grand Unified Boot Loader) under Linux. This is because CMS can be used to launch another guest operating system, but in doing so the CMS instance used ceases to exist.

Although CMS, as mentioned, increases functionality of the system, it still leaves much to be desired in some areas. Most notably when it comes to user-friendliness. CMS is completely console based, and contains no GUI as seen today. One might expect this to be like working with Linux through only command line, unfortunately this is not the case. The terminal will most likely feel old fashion and sluggish to use compared to a Linux shell as it lacks things like auto completion. Flexibility and freedom is often seen as one of the main advantages of terminals and shells. This can, however, quickly turn into a two-edged sword as it gives new and inexperienced users plenty of chances to cause serious damage to the system.

The CMS filesystem also bears mentioning. The CMS filesystem was designed to be fast, especially when dealing with file I/O operations. In contrast with files on Microsoft Windows or Linux, a CMS file is record oriented rather than byte oriented. Where a Linux file consists of a stream of bytes, a CMS file consists of a number of records (rows) of a specific length. As an example, a CMS file might be described as having a record length of 80 and 20 records. This means 20 rows of 80 characters, essentially amounting to 1600 bytes. This distinction is mostly important when it comes to transferring files between CMS and another operating system, since CMS files can not be read or written by another OS. The naming of files in CMS follows the syntax [file name] [file type] [file mode]; an example would be NOTES DATA A1. The file name can consist of up to eight alphanumeric character, and can be compared to the filename in Linux. The file type can also consist of up to eight alphanumeric character. It says something about the purpose of the file and can be compared to the file extension in Microsoft Windows. The file mode consists of one character and one number. The character represents the access character of the disk on which the file resides and can be compared to the partition character (like C:) in Microsoft Windows. The number ranges from 0 to 6 and is used by the system to decide how to treat the file. This number can be omitted in day-to-day activity.

Storage

When the term storage is used in mainframe context it usually refer to physical non-volatile storage like hard-drives. In IBM terms, hard-drives are referred to as Direct Access Storage Device or its acronym DASD (the physical DASDs are sometimes referred to as "real DASD"). In z/VM,

DASDs are normally segmented into 3390 disks (a former physical disk standard), but other models are used as well. These virtual DASDs are often referred to as "DASD packs" or "volumes". The 3390 disks can either be used as a whole, or it can be partitioned over several minidisks. Partitioning can be compared to partitioning a hard-disk in Microsoft Windows, and in z/VM each of the partitions are referred to as minidisks. Under z/VM each of these DASDs or minidisks can be shared among VMs, or be dedicated completely to a single VM. The size unit used when describing DASDs (and minidisks) are not bytes like in a personal computer, but rather cylinders. On the 3390 DASD, one cylinder is the equivalent to 849 960 bytes [3], or roughly 850 kB.

There are three types of 3390 DASDs in use today: 3390-3, 3390-9 and 3390-27. The 3390-3 is the most commonly used.

- 3390-3 (also known as mod 3) contains 3339 cylinders - about 3 GB in size.
- 3390-9 (also known as mod 9) contains 10017 cylinders - about 9 GB in size.
- 3390-27 (also known as mod 27) contains 30051 cylinders - about 27 GB in size.

Directory Maintenance Facility

The Directory Maintenance Facility will be referred to by the acronym DirMaint [4]. DirMaint is an IBM product designed to help with the management of the USER DIRECT file. The USER DIRECT file is where the guest definitions resides, it will be discussed to greater extent later in the text. Instead of editing USER DIRECT directly, commands are issued to DirMaint who then verifies the command and changes USER DIRECT. This reduces the chance of making errors, and makes sure the commands comes from authorized users. The main beneficial feature to this project however, is its ability to manage storage. Since guests created will be allocated storage in the form of cylinders in DASDs, it is important that this is handled properly. It is for instance vital that allocated cylinders do not overlap, as this would destabilize the system.

Programmable Operator Facility

The Programmable Operator Facility will be referred to by the acronym PROP [5]. PROP is designed to handle several tasks concerning system management. PROP is a service running under z/VM, meaning that it runs in a service virtual machine. It picks up messages sent to the VM it runs

in, and carries out actions depending on its configuration and the message. The three main features of PROP are:

1. Logging all messages that goes through PROP.
2. Routing messages to a predefined real user.
3. Executing code and commands depending on the incoming message.

The most important feature to this project is number three, Executing code and commands depending on the incoming message, as this also encompasses sending commands to different VMs. This makes it possible to verify the sender of the message, and check whether the sender is authorized for the requested action. It also makes it possible to control and limit the final target guest of the action. This is where much of the security of the system will reside.

2.4 IBMs current mainframe initiatives

IBM currently have two large scale projects of particular interest in the mainframe area. The first one centers on server consolidation, while the second one focuses on improving the user-friendliness of z/OS.

2.4.1 Server consolidation

The server consolidation initiative is part of IBMs project "Big Green". It is a five year plan started in 2007 to consolidate 3.900 distributed servers onto 33 Series z mainframes. By doing this, it is estimated to reduce the annual energy consumption by 80% and the total floor area used by 85%. This is made possible through the use of z/VM virtualization and z compatible Linux. It is an example of how a large scale virtualization environment implementation is necessary in a real world scenario, and with it comes the needs for a mean to consistently manage the environment.

2.4.2 z/OS simplification

The goal of the z/OS simplification initiative is to make the mainframe more user-friendly for non-experts. The initiative was started in 2006, and will be running for five years. During this timeperiod, IBM will invest approximately 100 million US dollars in making it easier to program, manage and administrate a mainframe system for the administrators and programmers. There will also be focus on increased automation when it comes to the development and deployment of applications on the mainframe environment. This shows IBMs interest and commitment in bringing the mainframe "to the people". At the end of the project, the power, potential and

utilization of the mainframe should be available to more than just the experts in the field.

2.5 MLN

MLN [6, 7] is an open source virtual machine administration tool. It was created by Kyrre Begnum at Oslo University College to improve the management aspect of a large number of virtual machines. Currently, it supports the popular virtualization platforms Xen [1] and User-Mode Linux [2].

MLN lets the administrator design and describe the VM setup in an easy to understand, declarative language. The language it is built up of blocks and attributes where the blocks are enclosed by curly brackets and each attribute normally have one value assigned. It also supports variables, making it easy to keep consistency and overview of the code. A pre-created template of the file system is then used to create the VMs. In MLN, logical groups of VMs are organized in projects. Every part of the project, the VMs and the network devices, are normally described in a single file. This makes it easy to administrate the project as a whole, letting the administrator (among other things) start, stop, create or destroy entire networks of VMs with a single command.

Two features of the MLN language that should be mentioned are superclasses and plug-ins. Superclasses should be familiar to anyone having experience in object oriented programming. Superclasses lets the administrator correlate repetitive, static attributes in a single block, and then let VMs in the project use the information through inheritance. This has several advantages, a few of which are mentioned here. It removes unnecessary redundant information, reduces the number of lines in the project file and reduces the number of (human) errors associated with repetitive tasks.

The second feature, plug-ins, make MLN remarkably flexible. The plug-in architecture allows MLN to use plug-ins written in the Perl programming language for mainly two purposes:

1. Introduce changes to the project before the project is built.

2. Add to the system configuration capabilities.

To further illustrate these features, an example is in order:

```
global {
  project example
  autoenum {
    superclass users
    addresses enum
    addresses_begin 2
    numhosts 30
    network 10.0.0.0
  }
  $gateway_address = 10.0.0.1
  $broadcast_address = 10.0.0.255
  $netmask = 255.255.255.0
}

superclass common {
  term screen
  xen
  lvm
}

superclass users {
  superclass common
  template ubuntu_user.ext3
  free_space 500M
  memory 128M
  network eth0 {
    netmask $netmask
    broadcast $broadcast_address
    gateway $gateway_address
    switch VirtualSwitch
  }
}

host gateway {
  superclass common
  template ubuntu_gw.ext3
  free_space 8GB
  memory 1024M
  network eth0 {
    netmask $netmask
    broadcast $broadcast_address
    address $gateway_address
    switch VirtualSwitch
  }
  network eth1 {
    netmask 255.255.255.0
    broadcast 192.168.1.255
    address 192.168.1.42
    gateway 192.168.1.1
  }
}

switch VirtualSwitch { }
```

The project starts with a `global` block which includes, among other things, the project name (example). This is the minimum requirement that all projects must contain. Further on in the global block the plug-in `autoenum` is used. The purpose of this plug-in is to automatically create a num-

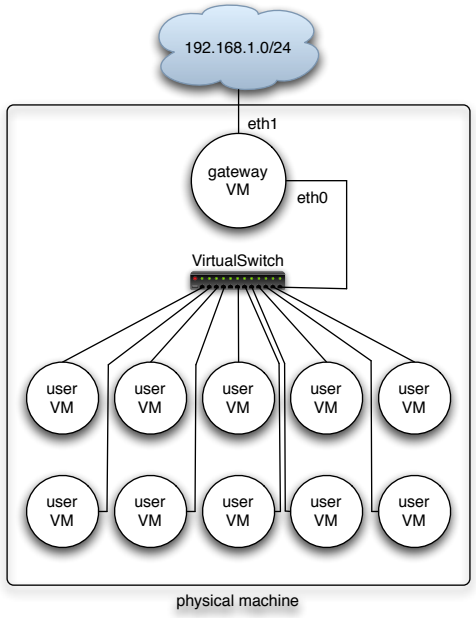

Figure 2.5: *Illustration of the end product (The number of users are scaled down).*

ber of identical VMs while at the same time giving each of them different IP addresses. The information contained in the autoenum block will be passed to the plug-in and used to create the VMs. The first line in the block, `superclass users`, lets the plug-in inherit the information contained in the superclass `user` further down in the file. It then declares that IP addresses should be assigned automatically starting with 2. Further, it specifies the number of VMs to create, and which network they will be part of. In this case, 30 VMs will be created in the network 10.0.0.0/24 using the IP range 10.0.0.2 – 10.0.0.31. This plug-in is a nice example of introducing changes to the project before the project is built. It also shows that scalability is not a problem. Without adding a single line of code, one can easily add 223 additional VMs simply by changing `numhosts 30` to `numhosts 253`.

After the autoenum block, the last part of the global block is used to demonstrate variables. The three critical IP addresses for the network are being contained in the three variables `$gateway_address`, `$broadcast_address` and `$netmask`. By using the variables through the file, the properties of the network can be changed by simply changing the values assigned to the variables. The modification will then propagate through the file the next time the project is built.

The next two blocks after the global block are examples of superclasses. The `common` superclass contains attributes that are intended for all the VMs

in this project, like which virtualization platform to use. The next superclass, `users`, will be used by the VMs auto-generated by autoenum (as instructed by the first line in the autoenum block previously discussed). `users` contain information like what pre-created template will be used and how much memory each VM will be assigned, Also note that it inherits the attributes from the `common` superclass. In the `network eth0` block, where the attributes for the VMs eth0 interface are defined, the system variables are used instead if static values. The final line in the `network eth0` block, `switch VirtualSwitch`, connects the VM to a central switch in the virtual network.

The next block, `host gateway`, shows how an individual VM is defined. It contains the same attributes as the `user` superclass, but with different values since greater hardware resources are necessary. A different template is also being used; this template would be designed to suite the needs of a gateway (i.e. contain a firewall and load balancing software). In addition to the `network eth0` block which connects it to the central switch in the 10.0.0.0/24 LAN, the gateway also has a second interface defined by the `network eth1` block. This is the interface that connects it to the outside network, making it capable of working as a gateway for the LAN. In the `network eth1` block, IP values have been assigned directly to the attributes. Although system variables of course could have been used, it was omitted to show this secondary approach.

At the end of the file the central switch, `switch VirtualSwitch`, is created.

The environment in which the VMs are to be created does not need to be one physical system. MLN allows it to be spread out over a number of physical machines so long as they are connected in a network and run the MLN network daemon. MLN also supports dynamic changing of the project properties like memory and disk space used, migrate VMs between servers and even change the virtualization platform. Although it is not one of MLNs main functions, it also provides some monitoring capabilities. This makes it easy for an administrator to see the projects, VMs and memory used on each of the physical machines.

MLN is currently the default management tool for virtual machines at Oslo University College and has also been used by University of Linköping, Sweeden and Oregon State University, US.

2.6 The viability of MLN on mainframes

Throughout this introduction chapter some points should have been made clear to attest the viability of MLN on mainframes. First of all, the capability

of MLN to ease the administration of VMs. What is especially important is its strength when it comes to scalability in administrating a high number of identical VMs. Secondly, mainframes suitability for hosting a high number of VMs. Due to the shear power of the mainframes, it is the ideal platform to consolidate and contain a large number of VMs. Third, the benefits of giving a Unix administrator a familiar environment. By abstracting z/VM through MLN it is possible for a Unix administrator already familiar with MLN to "hit the ground running" without acquiring a knowledgebase of z/VM. Fourth, z/VMs user-friendliness could generally be greatly improved for new and "casual" users, and the chance for accidental mistakes reduced.

Chapter 3

Approach

In this chapter, the design of the system will be decided. The first hands on experience with z/VM is described in section 3.1 and some examples are given to outline the look-and-feel of thef z/VM environment. In Chapter 3.2 the steps necessary to enable a Linux guest to communicate with CP is outlined and performed. The Programmable Operator is then presented and explained in section 4.1 followed by storage management in section 3.4. In this section the Directory Maintenance Facility will be presented and explained. Finally the networking possibilities are considered in section 4.3 before a quick description of the scenarios are given in section 4.6.

3.1 z/VM First Contact

The most common method used to access the z/VM environment on a mainframe is through the 3270 console. x3270 [8] is an IBM 3270 terminal emulator for the X Window System and Windows. It runs on most Unix-like operating systems and on Microsoft Windows. As one logs on to a z/VM environment on a mainframe, it is usual to be greeted by a login screen as seen in Figure 3.1. Through the execution of a `DIAL` command, access is gained to the specified second z/VM level as seen in Figure 3.2. When logging in to this environment with the system user MAINT, access is granted to CP. By executing the command `IPL CMS` (Initial Program Load Conversational Monitor System), the CMS operating system is loaded. From CMS it is possible to execute a wider specter of commands to examine or affect the system.

IBM have always placed great value on thorough documentation, the answer to most questions can be found in their multitude of books and papers. In fact, some answers appear difficult to find because of the abundance of information. Several RedBooks have been found that will be used and referred to during this project. Especially those documenting all avail-

able CP and CMS commands are predicted to be of immense help and will probably be consulted frequently:

- z/VM CP Commands and Utilities Reference [9]
- z/VM CP Planning and Administration [10]
- z/VM Directory Maintenance Facility Commands Reference [11]
- z/VM Directory Maintenance Facility Tailoring and Administration Guide [12]
- Program Directory for IBM z/VM Directory Maintenance Facility Feature [13]

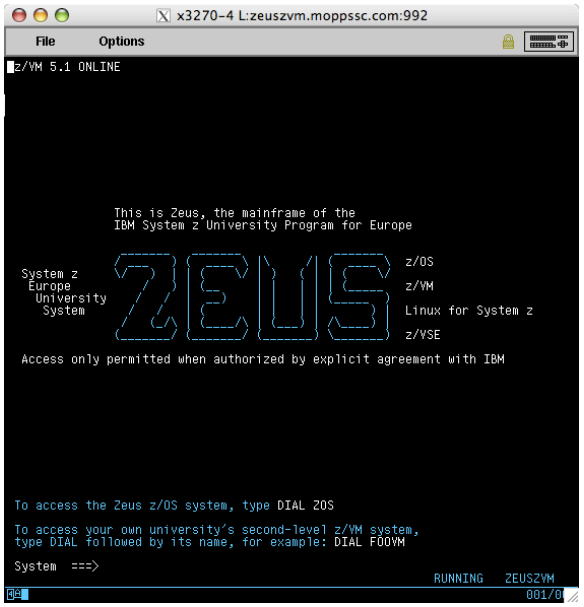

Figure 3.1: *The login screen on the Zeus mainframe.*

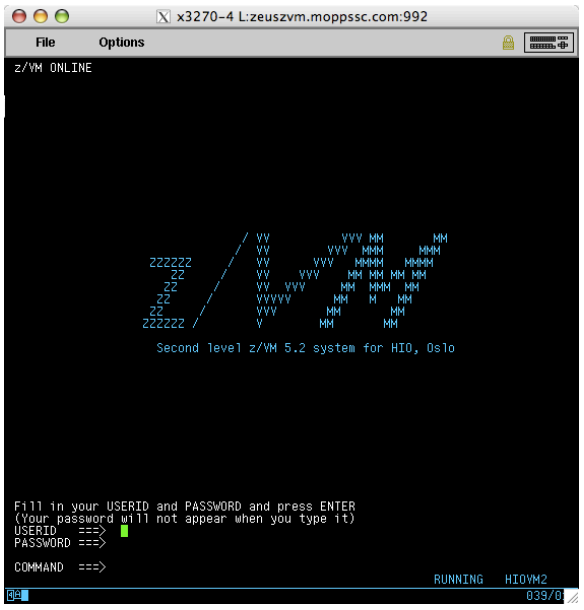

Figure 3.2: *The login screen for the dedicated second z/VM level on the Zeus mainframe.*

The `LISTFILE` command lists the files in the specified minidisk. If no minidisk is specified, it will automatically list the files in the MODE A minidisk. The A minidisk can be compared to the `/home/user` folder in Linux.

```
                                  Fullscreen CMS         Lines 96 - 120 of 120
                                                          Columns 1 - 79 of 81
  Ready; T=0.01/0.01 15:17:05
  CMS
  listfile
  $$NLS$   TEXT     A2
  $VMFBLD  $MSGLOG  A1
  DMSAMENG LANGMAP  A5
  DMSGER   LANGMAP  A5
  DMSKANJI LANGMAP  A5
  DMSUCENG LANGMAP  A5
  LASTING  GLOBALV  A1
  MBFIND   XEDIT    A1
  MDISKMAP EXEC     A1
  MDISKMAP XEDIT    A1
  NLSAMENG DCSSMAP  A5
  NLSGER   DCSSMAP  A5
  NLSHDR   LISTING  A1
  NLSKANJI DCSSMAP  A5
  NLSUCENG DCSSMAP  A5
  PK       EXEC     A1
  PROFILE  EXEC     A1
  PROFILE  XEDIT    A1
  SETUP    $LINKS   A1
  SYN      SYNONYM  A1
  USER     MDISKMAP A1
  Ready; T=0.01/0.01 15:17:44




  PF1=Help      2=Pop_Msg   3=Quit       4=Clear_Top  5=Filelist   6=Retrieve
  PF7=Backward  8=Forward   9=Rdrlist   10=Left      11=Right     12=Cmdline
  ====>
 15:17:44                              Enter a command or press a PF or PA key
================================================================================
```

As can be seen in the printout, the files are listed in the previously mentioned syntax [file name] [file type] [file mode]. File types in CMS are not strict, but there are common ways to designate filetypes. For example, EXEC usually means an executable file while XEDIT is usually a text file.

A series of `QUERY` commands can be executed to learn more about the environment. Here an example of the `QUERY DISK` which lists all the accessed disks.

```
                                  Fullscreen CMS        Lines 109 - 120 of 120
                                                          Columns 1 - 79 of 81
  Ready; T=0.01/0.01 15:41:41
  query disk
  LABEL  VDEV M  STAT   CYL TYPE BLKSZ   FILES  BLKS USED-(%) BLKS LEFT  BLK TOT
  MNT191 191  A   R/W   175 3390 4096       21        132-01      31368      315
  MNT5E5 5E5  B   R/W     9 3390 4096      132       1284-79        336       16
  MNT2CC 2CC  C   R/W     5 3390 4096       58        423-47        477        9
  MNT51D 51D  D   R/W    13 3390 4096      249       1126-48       1214       23
  MNTCF1 CF1  F   R/O   120 3390 4096       14       3296-15      18304      216
  MNT190 190  S   R/O   100 3390 4096      687      14517-81       3483      180
  TCM592 592  T   R/O    67 3390 4096      885       8496-70       3564      120
  MNT19E 19E  Y/S R/O   250 3390 4096     1063      27496-61      17504      450
  Ready; T=0.01/0.01 15:43:10

  PF1=Help      2=Pop_Msg   3=Quit       4=Clear_Top  5=Filelist   6=Retrieve
  PF7=Backward  8=Forward   9=Rdrlist   10=Left      11=Right     12=Cmdline
  ====>
 15:43:10                              Enter a command or press a PF or PA key
================================================================================
```

This list gives a lot of useful information. It shows the disk label, virtual device address, mode, read/write access, cylinders on disk, disk type and information about how much of the disk is utilized.

By using the XAUTOLOG and SIGNAL SHUTDOWN commands, VMs can be started and stopped. It is shown here in conjunction with the QUERY NAMES command whose output shows all the running guests (VMs) on the system.

```
                                Fullscreen CMS           Lines 49 - 74 of 74
                                                        Columns 1 - 79 of 81
  CMS
  query names
  LINUX2   - DSC , DTCVSW2  - DSC , DTCVSW1  - DSC , OPERSYMP - DSC
  DISKACNT - DSC , EREP     - DSC , OPERATOR - 0009, MAINT    - 0020
  Ready; T=0.01/0.01 14:18:46
  xautolog linux1
  Command accepted
  Ready; T=0.01/0.01 14:18:59
  AUTO LOGON  ***       LINUX1   USERS = 9
  HCPCLS6056I XAUTOLOG information for LINUX1: The IPL command is verified by th
 IPL command processor.
  CMS
  query names
  LINUX2   - DSC , DTCVSW2  - DSC , DTCVSW1  - DSC , OPERSYMP - DSC
  DISKACNT - DSC , EREP     - DSC , OPERATOR - 0009, LINUX1   - DSC
  MAINT    - 0020
  Ready; T=0.01/0.01 14:19:52
  signal shutdown linux1
  Ready; T=0.01/0.01 14:20:07
  HCPSIG2113I User LINUX1 has reported successful termination
  USER DSC   LOGOFF AS  LINUX1   USERS = 8     AFTER SIGNAL
  CMS
  query names
  LINUX2   - DSC , DTCVSW2  - DSC , DTCVSW1  - DSC , OPERSYMP - DSC
  DISKACNT - DSC , EREP     - DSC , OPERATOR - 0009, MAINT    - 0020
  Ready; T=0.01/0.01 14:20:56

  PF1=Help      2=Pop_Msg   3=Quit       4=Clear_Top  5=Filelist   6=Retrieve
  PF7=Backward  8=Forward   9=Rdrlist   10=Left      11=Right     12=Cmdline
  ====>
 14:20:56                             Enter a command or press a PF or PA key
================================================================================
```

The QUERY NAMES command lists the running VMs. In the first query, only LINUX2 of the Linux VMs are running, but no terminal is connected to it (it is listed with DSC). By executing XAUTOLOG LINUX1 the LINUX1 VM is started. After the VM has been given time to boot (30-40 seconds in this case), it can be seen by running QUERY NAMES again. Through the SIGNAL SHUTDOWN LINUX1 command, a signal is sent to the LINUX1 VM telling it to shut down. After allowing the VM to shut down (about 30 seconds in this case), the QUERY NAMES shows that it is no longer running.

The VMs themselves are defined in the `USER DIRECT` file. A small portion of a `USER DIRECT` file is shown here through the `XEDIT` text editor. It lists the definition for three Linux guests.

```
 USER     DIRECT   C1  F 80  Trunc=72 Size=1704 Line=1679 Col=1 Alt=0

01679 *
01680 * Linux guests for HIO
01681 USER LINUX1 LBYONLY 256M 512M G
01682   INCLUDE ZEUSCMS
01683   LOGONBY MAINT
01684   DATEFORMAT FULLDATE
01685   OPTION TODENABLE
01686   NICDEF 700 TYPE QDIO LAN SYSTEM LOCALNET
01687   LINK MAINT 1000 191 RR
01688   MDISK 0100 3390 0002 3330 USER01 MR
01689 USER LINUX2 LBYONLY 256M 512M G
01690   INCLUDE ZEUSCMS
01691   LOGONBY MAINT
01692   DATEFORMAT FULLDATE
01693   OPTION TODENABLE
01694   NICDEF 700 TYPE QDIO LAN SYSTEM LOCALNET
01695   LINK MAINT 1000 191 RR
01696   MDISK 0100 3390 0002 3330 USER02 MR
01697 USER LINUX3 LBYONLY 256M 512M G
01698   INCLUDE ZEUSCMS
01699   LOGONBY MAINT
01700   DATEFORMAT FULLDATE
01701   OPTION TODENABLE
01702   NICDEF 700 TYPE QDIO LAN SYSTEM LOCALNET
01703   LINK MAINT 1000 191 RR
01704   MDISK 0100 3390 0002 3330 USER03 MR
01705 * * * End of File * * *

====>
================================================================================
```

This printout needs a somewhat more detailed description. The LINUX1 will be examined in this case, and an explanation of each line will be given.

```
USER LINUX1 LBYONLY 256M 512M G
```

This is the first line of the VM configuration. It defines the user `LINUX1` with the password `LBYONLY`. The `LBYONLY` is not the actual password, but rather a reserved argument for the password field. Its effect is that to log

on to this guest with the LOGON command, the BY option must be used. Also, the user ID cannot be used to log on to guests with the BY option, or in fact perform any function that requires a password. In effect, this user ID does not actually have a password. The user will run in a VM with 256 MB (`256M`) of memory. The user will have the possibility to expand this to 512 MB (`512M`) if necessary. The `G` at the end of the line represents the CP privilege class that the user will run under. The default CP privilege classes ranges from A to G, where A represents the highest administrator while G represents a normal user.

```
INCLUDE ZEUSCMS
```

This `INCLUDE` statement imports the already defined `ZEUSCMS` profile. In this case, the profile is defined higher in the `USER DIRECT` file:

```
1  00083 PROFILE ZEUSCMS
2  00084   MACHINE ESA
3  00085   IPL CMS PARM AUTOCR
4  00086   SPOOL 000C 2540 READER *
5  00087   SPOOL 000D 2540 PUNCH A
6  00088   SPOOL 000E 1403 A
7  00089   CONSOLE 009 3215 T
8  00090   LINK MAINT 0190 0190 RR
9  00091   LINK MAINT 019D 019D RR
10 00092   LINK MAINT 019E 019E RR
11 00093   LINK TCPMAINT 0592 0592 RR
```

Since the profile is used in the definition of the `LINUX1` VM, it will be explained before continuing on the `LINUX1` block.

```
PROFILE ZEUSCMS
```

The first line defines the block entry as a `PROFILE` with the name `ZEUSCMS`.

```
MACHINE ESA
```

This line sets the machine architecture the VM will simulate. The ESA architecture is chosen here, other valid choices are XA and XC.

```
IPL CMS PARM AUTOCR
```

This line makes the VM automatically boot CMS when the VM starts (`IPL CMS`). The `PARM AUTOCR` simulates the pressing of ENTER as the input to the VM at the initial VM READ. The PROFILE EXEC will be executed automatically if it exists in mode A.

```
SPOOL 000C 2540 READER *
```

Defines a virtual card reader for the VM. Card Readers and punchers have not been discussed so far, but they are commonly used to transfer data between VMs.

```
SPOOL 000D 2540 PUNCH A
```

Defines a virtual card puncher for the VM.

```
SPOOL 000E 1403 A
```

Defines a virtual printer for the VM. A virtual printer might sound strange, but it can for instance be connected to a printer server to facilitate actual printing.

```
CONSOLE 009 3215 T
```

Defines the type of virtual I/O support CP provides for the display and its virtual address.

```
LINK MAINT 019x 019x RR
```

Links to a disk owned by another guest. In this case the disk belongs to MAINT. The two hexadecimal numbers are the disks virtual address on MAINT and the virtual address it will be linked to on the created guest respectively. RR means that the disk will be Read Only.

That marks the end of the `PROFILE ZEUSCMS` block, and we continue with the `USER LINUX1` block.

```
LOGONBY MAINT
```

Makes it possible for the user `MAINT` to log on to the guest with MAINTs own password using LOGON with the BY option.

```
DATEFORMAT FULLDATE
```

Sets the default date format for the guest.

```
OPTION TODENABLE
```

Makes it possible for the user to change the time and date for the guest.

```
NICDEF 700 TYPE QDIO LAN SYSTEM LOCALNET
```

Defines a virtual network interface of type Queued Direct Input/Output (QDIO) and connects it to the virtual switch LOCALNET.

```
LINK MAINT 1000 191 RR
```

As previously explained in the PROFILE ZEUSCMS block.

```
MDISK 0100 3390 0002 3330 USER01 MR
```

Defines a new minidisk to be owned by this guest. The first number (0100) is the virtual device number while the second (3390) is the device type. The next three parameters (0002 3330 USER01) says that the minidisk starts on cylinder 0002 and allocates 3330 cylinders on the real DASD volume USER01. MR sets the minidisk to multiple-write access.

3.2 CP interaction through Linux / Managing VMs from Linux

z/VM allows commands to be issued from a running Linux guest to CP. In this case this happens through vmcp, a kernel module in Linux that handles the communication. The module is included in SUSE Linux Enterprise Server 10 (SLES10) which is used in this environment, but it is not activated by default. Although the module can be loaded manually by running modprobe vmcp, it is desirable to have it load on startup as default on the VM LINUX1. To do this, it is necessary to ssh into LINUX1 and edit the file /etc/sysconfig/kernel . vmcp is simply added to MODULES_LOADED_ON_BOOT as shown below:

```
1 ## Type:               string
2 ## ServiceRestart:     boot.loadmodules
3 #
4 # This variable contains the list of modules to be loaded
5 # once the main filesystem is active
6 # You will find a few default modules for hardware which
7 # can not be detected automatically.
8 #
9 MODULES_LOADED_ON_BOOT="vmcp"
```

After saving the changed file, restart LINUX1 through CP (from the normal x3270 console). It is now possible to issue CP commands from the command line in LINUX1 by using the vmcp command.

```
hiovm2-linux1:~ # vmcp query names
MAINT    - 0020, LINUX2   - DSC , DTCVSW2  - DSC , DTCVSW1  - DSC
OPERSYMP - DSC , DISKACNT - DSC , EREP     - DSC , OPERATOR - 0009
LINUX1   - DSC
hiovm2-linux1:~ #
```

As seen the output from CP is sent back to STDOUT on LINUX1. What commands can be executed is determined by the privilege class the VM is running in. `QUERY NAMES` is a command that can be issued by every standard privilege class, and so it was executed successfully. When trying to run a higher level command, it becomes clear that LINUX1 runs with restricted access.

```
hiovm2-linux1:~ # vmcp xautolog linux3
HCPLGA6050E Your userid is not authorized to automatically logon userid LINUX3
Error: non-zero CP response for command 'XAUTOLOG LINUX3': #6050
hiovm2-linux1:~ #
```

When trying to start the LINUX3 VM, CP responds that the used userid is not authorized to automatically logon userid LINUX3. With a quick query check, it becomes clear why.

```
hiovm2-linux1:~ # vmcp query privclass
Privilege classes for user LINUX1
Currently: G
Directory: G
The privilege classes are not locked against changes.
hiovm2-linux1:~ #
```

LINUX1 runs under privilege class G (normal user). To be able to perform administrative tasks like `XAUTOLOG` and `SIGNAL SHUTDOWN` it is necessary to run under a higher privilege class (A, B or C). The simplest solution to this would be to change the privilege class for LINUX1 in USER DIRECT from G to A. This would give LINUX1 the necessary privileges to administrate other guests, and therein lies the problem. This solution would give LINUX1 privileges to control ALL guests under z/VM, not just the Linux guests that it is in charge of but also guests used by z/VM to function properly (Figure 3.3). This would present a too great security risk, and is therefor an unacceptable solution.

Another alternative would be to create a special privilege class for the administrating Linux guest where the necessary CP commands are added. This would not really be a noticeable improvement form the previously mentioned solution, as all the high-risk commands (like `SIGNAL SHUTDOWN`) would have to be added to the new class (Figure 3.4).

The third solution is to create a proxy guest for security purposes only. The proxy would be a service virtual machine running PROP [4] (programmable operator). All commands from the administrating Linux (LINUX1) guest to other guests would be sent to this proxy instead. LINUX1 would still run under privilege class G, but the proxy will run under privilege class A. The proxy would then receive the command with some additional parameters like which guest sent the command and which guest was the target. The proxy checks if the sender is allowed to issue the command, but more importantly that it is allowed to send to the target. The target name is checked against a file on the proxy containing the names of the restricted guests, and if it does not find a match then the target is valid.

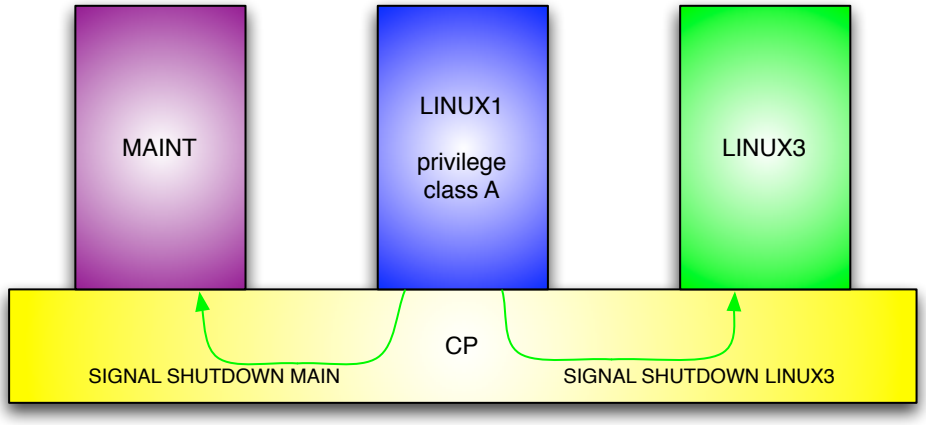

Figure 3.3: *Privilege class A LINUX1. LINUX1 is able to shut down LINUX3, but it is also able to shut down MAINT. This is a very bad thing as it does not protect the integrity of the system at all.*

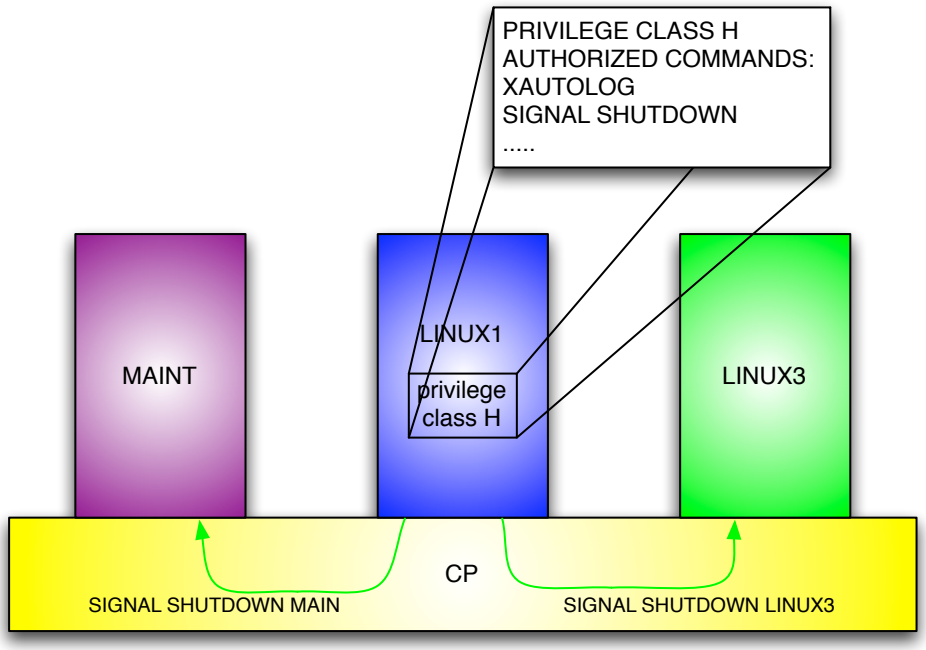

Figure 3.4: *LINUX1 running under custom created H privilege class. In this case, nothing has really changed from giving LINUX1 privilege class A. LINUX1 still has the possibility to destabilize the system severely.*

3.3 Programable Operator Facility

PROP is the solution chosen for this project. The reason being as explained the increased security. It can be argued that this increases the complexity of the system, and increases the knowledge requirement for setting up the system. This is indeed true, but it can also be seen from a slightly different perspective. The value of the finished product diminishes greatly if nobody is willing to use it because of the security risk, security is after all a very important aspect of virtualization and the mainframe. As for the increased knowledge requirement in setting up the system, this is only for the installation phase. For a system like the mainframe, whose operation time span can easily reach 10 years, this was seen as an acceptable tradeoff. After all, z/VM knowledge was a requirement for the installation process from the very beginning. This will however not increase the requirements for using the established system.

As seen in Figure 3.5, LINUX1 issues a command to shut down the LINUX3 guest. Instead of sending the command to LINUX3, in which case it would be rejected because LINUX1 only runs under privilege class G, the command is sent to PROP. When PROP receives the command it runs a number of checks to verify that this is a legal action. It will check that LINUX1 is allowed to execute this command, and that the target in not on a list containing the names of guests not to be administrated by LINUX1. If all checks are cleared, PROP will send the command to LINUX3. Because PROP runs under privilege class A, the command will be accepted and executed.

In Figure 3.6, LINUX1 tries to issue the SIGNAL SHUTDOWN command on MAIN. This would be an undesirable action since MAIN is not a linux guest and is outside the jurisdiction of LINUX1. The command is sent to PROP, and PROP runs through the checks to clear the command. PROP sees that LINUX1 is indeed allowed to issue the SIGNAL SHUTDOWN command. However, when checking the validity of the target PROP finds the guest name in the list of illegal targets. Because of this, PROP drops the command and the integrity of the system remains uncompromised.

3.4 Storage Management

DirMaint is certainly not the perfect solution for this project when it comes to the choice of storage management solution. First of all, it is yet another underlying subsystem. Secondly, it is not a free IBM product. It is shipped with z/VM, but it is not activated unless paid for. There are however several reasons favoring this solution. The main reason is perhaps the simplest: z/VM storage administration is not what this project is about. Al-

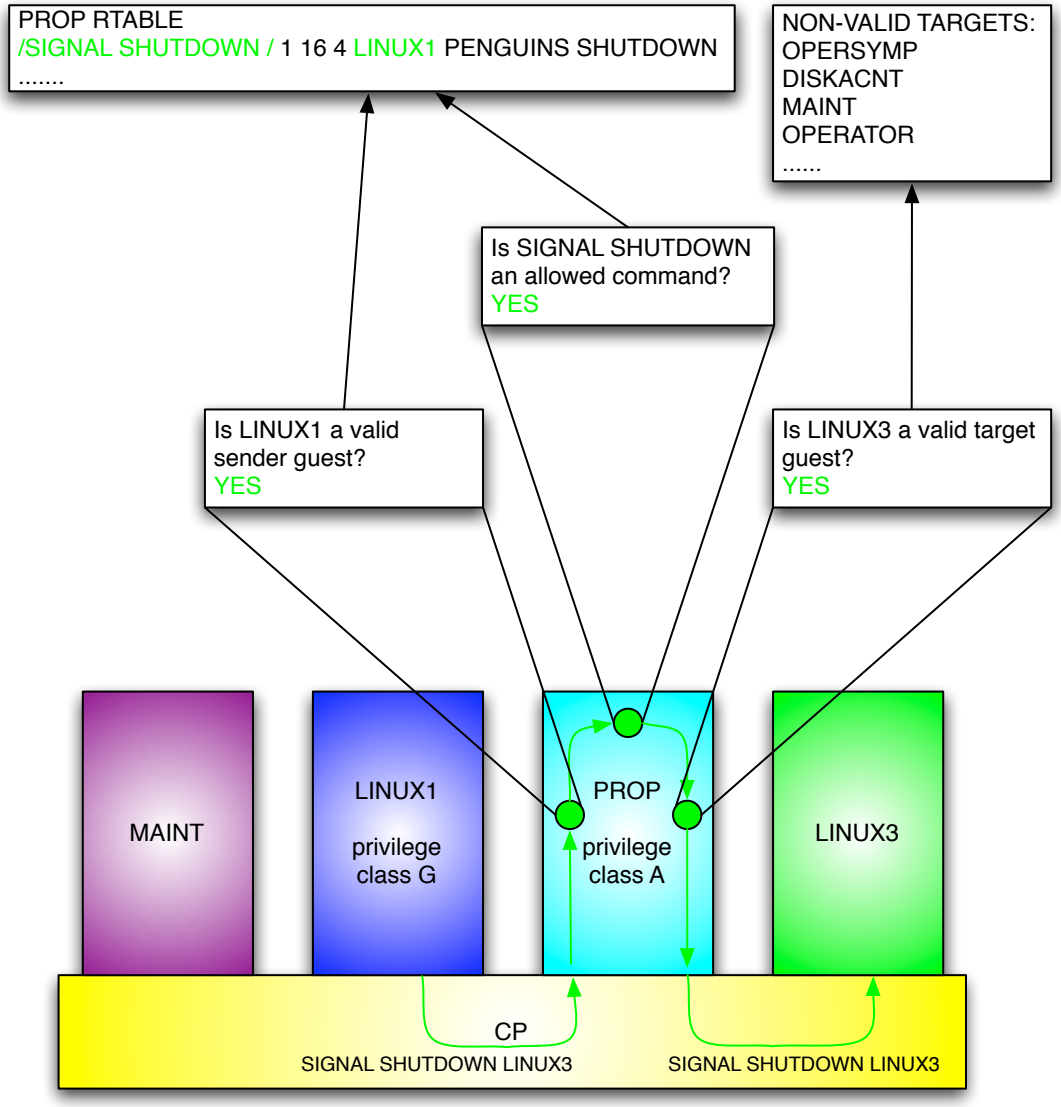

Figure 3.5: *LINUX1 executes a legal command on a legal target guest. The command goes to PROP who runs it through several checks. Since everything seems to be in order, PROP executes the command on the target guest with the necessary privilege class.*

though it would certainly be interesting to create a module that handled storage allocation, it is not a priority at this time. Which brings up another reason, time. The time span of this project is quite limited, and so there is simply not the time to create a satisfactory alternative to DirMaints functionality. As mentioned when first introducing DirMaint, storage allocation

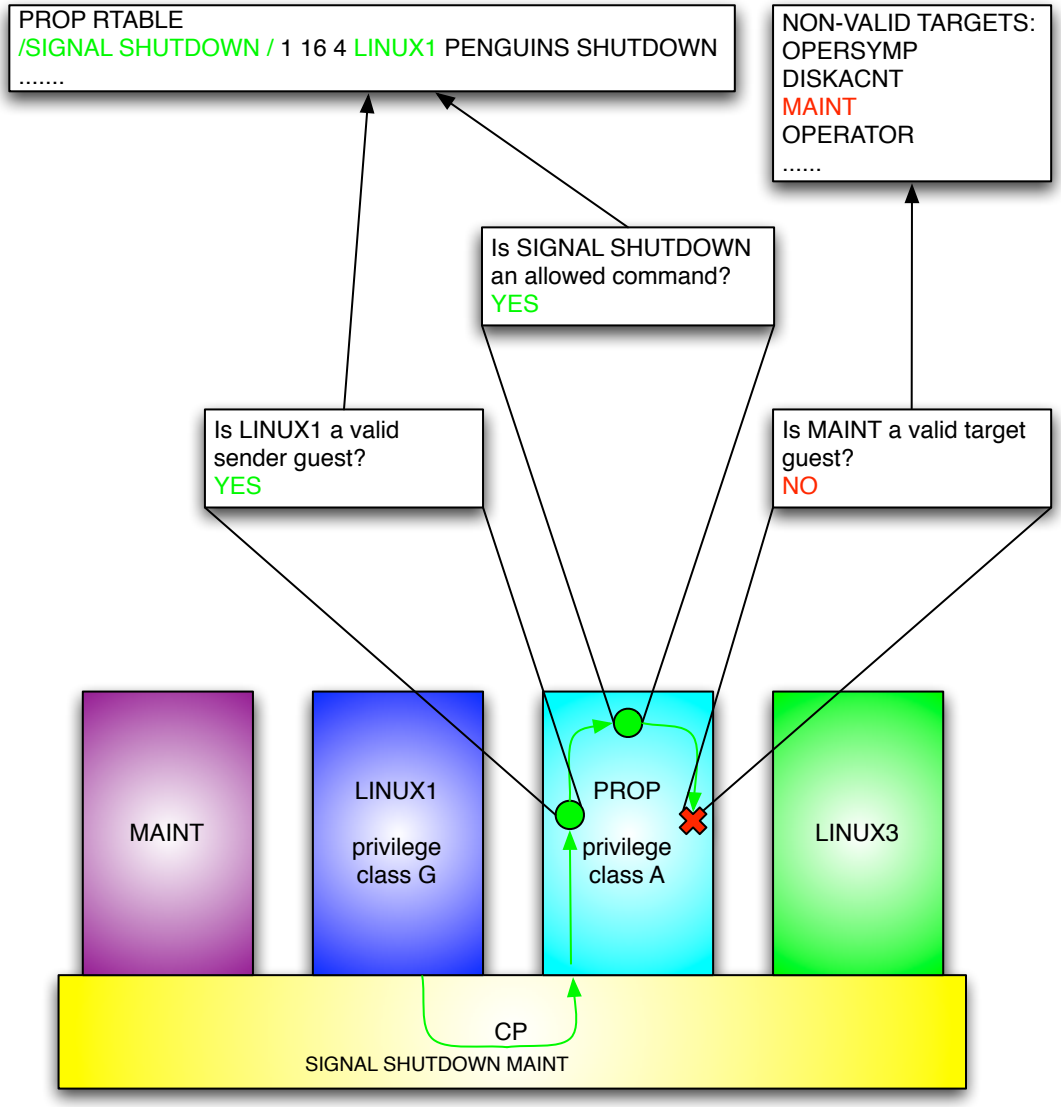

Figure 3.6: *LINUX1 executes a legal command on an illegal target guest. The command goes to PROP who runs it through several checks. Everything seems to be in order until PROP checks the validity of the target. The target guest, MAIN, is on PROPs list of non-valid targets and so the command is discarded as illegal.*

is extremely delicate and it would take up to much resources to create a system with the needed reliability and stability.

Other alternatives was of course also considered. A module to handle the task would have to have a database with DASD resources made available

to the system. It would then need to keep track of which cylinders was allocated, and update this information when changes to guest configuration were made and when guests were created and destroyed. This system would run on the administrating Linux guest, and the database containing the DASD resources would have to be managed manually or fetch data from a file in z/VM with regular intervals (in which case the file on z/VM would have to be managed manually).

The system could alternatively reside in CMS, but it would then have to use flat files instead of databases. This would also increase the communication between the administrating Linux guest and CP considerably since the Linux guest would have to consult the flat files for every storage related change that was made. This approach would however give access to handy features like DISKMAP (creates a report on disk cylinder use).

As mentioned earlier in this text, DirMaint does more than just handle the allocation of storage for guests. When installed and activated it takes control of the entire working of USER DIRECT. This is not optional, an so all operation that manipulate the configuration of guests, from creation to destruction, must go through DirMaint.

DirMaint consists of mainly four service VMs.

DIRMAINT
: The Directory Maintenance Service VM is the primary server. It holds and manipulates the USER DIRECT (actually called the source directory in DirMaint) file, validates the input it receives and checks that the input came from an authorized administrator. It is given control over a pool of DASD storage (cylinders) and is tasked with the allocation of storage to guests. Finally, it controls the other servers that comprises the Directory Maintenance Facility.

5VMDIR10
: The 5VMDIR10 service VM is DirMaints install and service server. It owns the DASD space containing all the base code shipped with DirMaint in addition to optional source files, sample files, help files and files customized by the user. It also owns disks used by DirMaint for testing and production purposes.

DATAMOVE
: The DATAMOVE service VM handles DASD manipulation on behalf of DirMaint. More specifically it has mainly three tasks. The first task is to formate newly allocated DASD space for other guests. The second is to copy or move data from one disk to another. The third is to formate deallocated disk space to prevent the content from being exposed to the next user to access the disk area. There may be more

than one DATAMOVE service VM working on a system to share the workload.

DIRMSAT
: The Cluster Satellite Synchronization Service VM is used in multi-system clusters. One DIRMSAT runs on each system where it controls its respective systems USER DIRECT. The DIRMSAT servers are then used by the DIRMAINT server to propagate changes through the cluster. The DIRMSAT servers will receive instructions from DIRMAINT to change USER DIRECT, thereby synchronizing it with the central USER DIRECT on the DIRMAINT server.

3.5 Network

To support the possibility to network the VMs created through MLN, a virtual networking technology must be chosen. There are several possibilities offered in z/VM. They all have their respective areas of use, and it is necessary to decide which solution is best for implementation as MLNs underlying network technology on z/VM.

A term that should be explained before looking at some of the options is OSA-Express. OSA stands for Open Systems Adapter and is a physical network controller. The adapter incorporates several hardware features and supports several network transport protocols, some being fast Ethernet, gigabit Ethernet, 10 gigabit Ethernet, token ring and ATM.

Also, QDIO stands for Queued Direct Input/Output. It is a highly efficient data transfer interface mechanism. It makes it possible to buffer data directly in the hosts main storage, thereby bypassing several steps of the I/O process.

As a final note, the TCP/IP stack in z/VM is actually running in a dedicated service VM called TCPIP.

3.5.1 HiperSockets

HiperSockets is an IBM technology providing high speed TCP/IP communication. It requires no physical network devices as it exists exclusively in memory. This also accounts for its high transfer speed. HiperSockets are usually deployed between LPARS, but can also be created and used in a single LPAR. This seems like an ideal solution, but it has one crucial limitation: There can only be defined 16 HiperSockets on a system. This makes it unsuitable for use in MLN.

3.5.2 Guest LAN

A more appropriate name for Guest LAN would have been Virtual LAN had the term not already been well established in the network area. A Guest LAN, introduced in z/VM 4.2, is a virtualized LAN segment. No physical networking equipment is required and there is no limitation on how many Guest LANs can be defined at the same time. The LAN segment itself cannot be connected to a physical network devise. If the segment is to have contact with the "outside world" then this must happen through a dedicated gateway VM connected to both the LAN segment and a physical device as shown in Figure 3.7.

There are two different types of Guest LANs available [14]:

- QDIO which emulates a OSA-Express device
  - IPv4 and IPv6 support
  - Easy to migrate from QDIO Guest LAN to VSWITCH
  - Ethernet transport
  - Asynchronous
  - Can be used as an OSA-Express test network
- Internal QDIO (iQDIO) which emulates a HiperSocket connection
  - IPv4 support
  - Supports multicast router connections
  - Deploy MTUs larger than 8 K
  - Synchronous
  - Can be used as a HiperSockets test network
  - Slightly smaller path length in CP than QDIO Guest LAN

3.5.3 Virtual Switch

The Virtual Switch, or VSWITCH, is the newest form of virtualized LAN available to z/VM. Introduced in z/VM 4.4 it builds on the already existing Guest LAN technology. Unlike a Guest LAN, the VSWITCH can be connected directly to a physical OSA-Express port (Figure 3.8). In fact, a VSWITCH can be connected to up to three separate OSA-Express ports. The two extra ports would then work as backup ports. The guests connected to the VSWITCH resides on the same subnet as the physical port the VSWITCH is connected to. It should be noted that a VSWICH does not have to be connected to a physical device, it would then be compartmentalized to the guests connected to it.

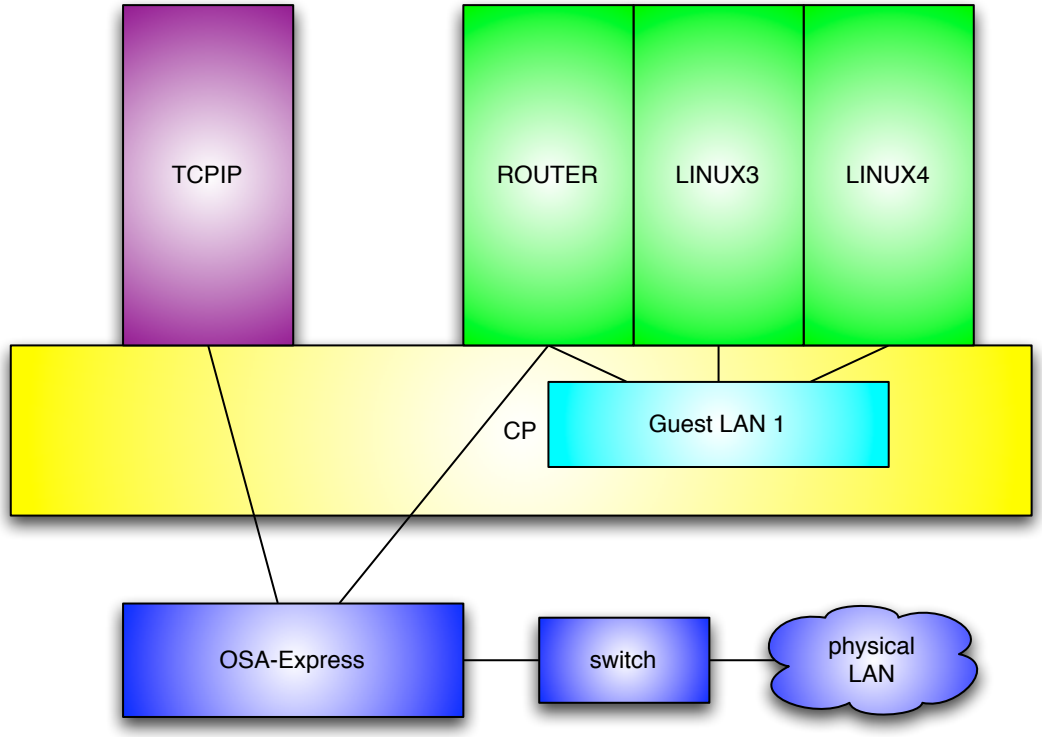

Figure 3.7: *Architecture of a Guest LAN network. The Guest LAN is able to connect the guest together in a network, but is unable to provide access to a real OSA-Express device. Therefore a guest router gateway* must *be in place for the network to gain access to the physical LAN.*

Although the VSWITCH has removed the need for router guests, another necessity has been introduced. z/VM requires that the OSA-Express devise used by the VSWITCH must be owned by a guest. These guests are called VSWITCH controllers and are basically extra TCP/IP stacks that manage the OSA-Express on behalf of the systems connected to the VSWITCH. By default, each z/VM environment comes with two VSWITCH controllers, DTCVSW1 and DTCVSW2, but more can be added manually if more redundancy is required.

A VSWITCH can be created in two ways. It can be created statically in the `SYSTEM CONFIG` file, in which case the VSWITCH will be created at system start up or when the system is told to reread the `SYSTEM CONFIG` file. The second method is to create it dynamically using the `DEFINE VSWITCH`, in this case the VSWITCH is created instantly without the need to reread the system configuration settings. However, this makes the dynamically created VSWITCH volatile so in the case of a system restart or shutdown

the VSWITCH would cease to exist.

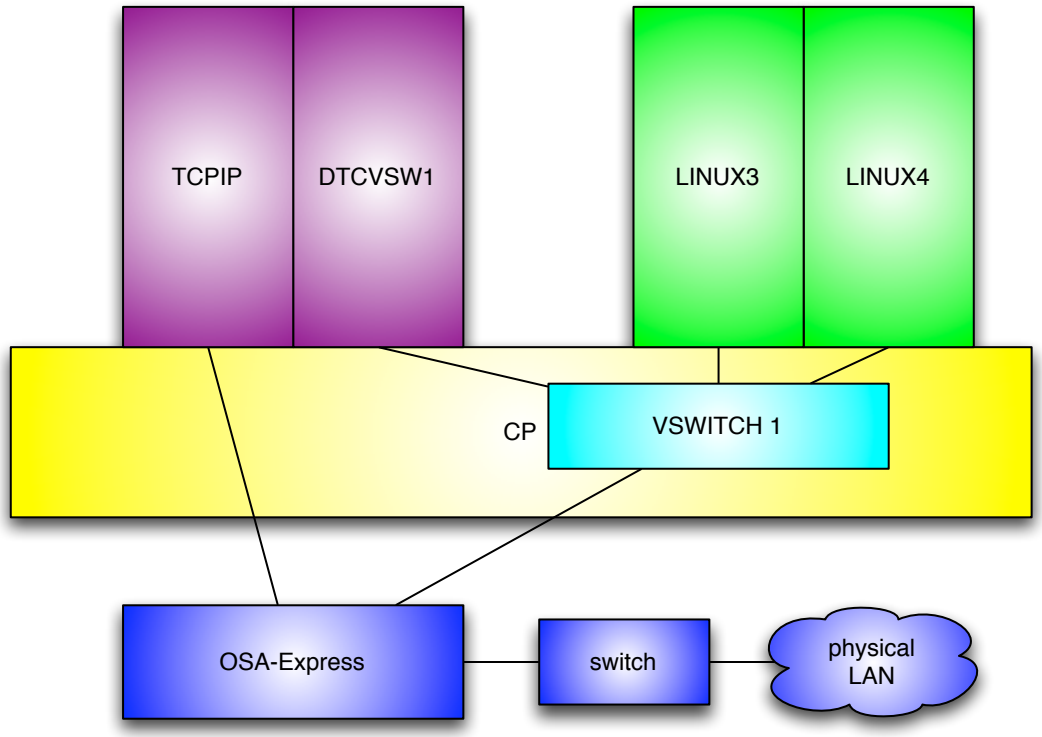

Figure 3.8: *Architecture of a Virtual Switch network. The VSWITCH connects all the guests in a network and also has the option of being directly connected to a OSA-Express device and therefor the physical LAN. No guest router gateway is necessary. (Note that the connection to DTCVSW1 is automatically handled by z/VM)*

When the VSWITCH was introduced in z/VM 4.4 it only operated on OSI layer 3. This means that it only routes based on IP addresses without considering the MAC address. This effectively limits its use to TCP/IP applications. For all communication with the physical LAN, the MAC address of the OSA-Express port is used. Thus when a guest on the VSWITCH wants to send a packet to the physical part of the LAN, the packet is encapsulated by an Ethernet frame with the OSA-Express port MAC address as the source. When packets come from the physical LAN with the OSA-Express port MAC address as the destination, the OSA-Express device simply strips away the Ethernet frame and sends the remaining part to the VSWITCH.

In z/VM 5.1 the possibility to operate on OSI layer 2 was introduced to the VSWITCH in addition to its already existing layer 3 support. This

means that it more closely lives up to its name as it can virtualize a normal, fully functional physical layer 2 switch (layer 3 switches are usually refereed to as routers). In layer 2 mode the switch sends and receives Ethernet framed packets using MAC addresses. This greatly increases the usefulness of the VSWITCH as its use is no longer limited to TCP/IP applications.

Since MLN uses layer 2 switches, the VSWITCH was chosen as the underlaying z/VM networking technology. Also, IBM generally recommends to use the VSWITCH as opposed to Guest LAN whenever possible.

Figure 3.9 shows an example of an MLN environment where different network solutions have been implemented for the different guest groups. For simplicities sake, TCPIP and the VSWITCH controller was omitted from the diagram. The environment has one statically defined VSWITCH, VSWMLN, that works as the environments gateway, all other VSWITCHes are defined dynamically by MLN (in this case VSWG1 and VSWG2).

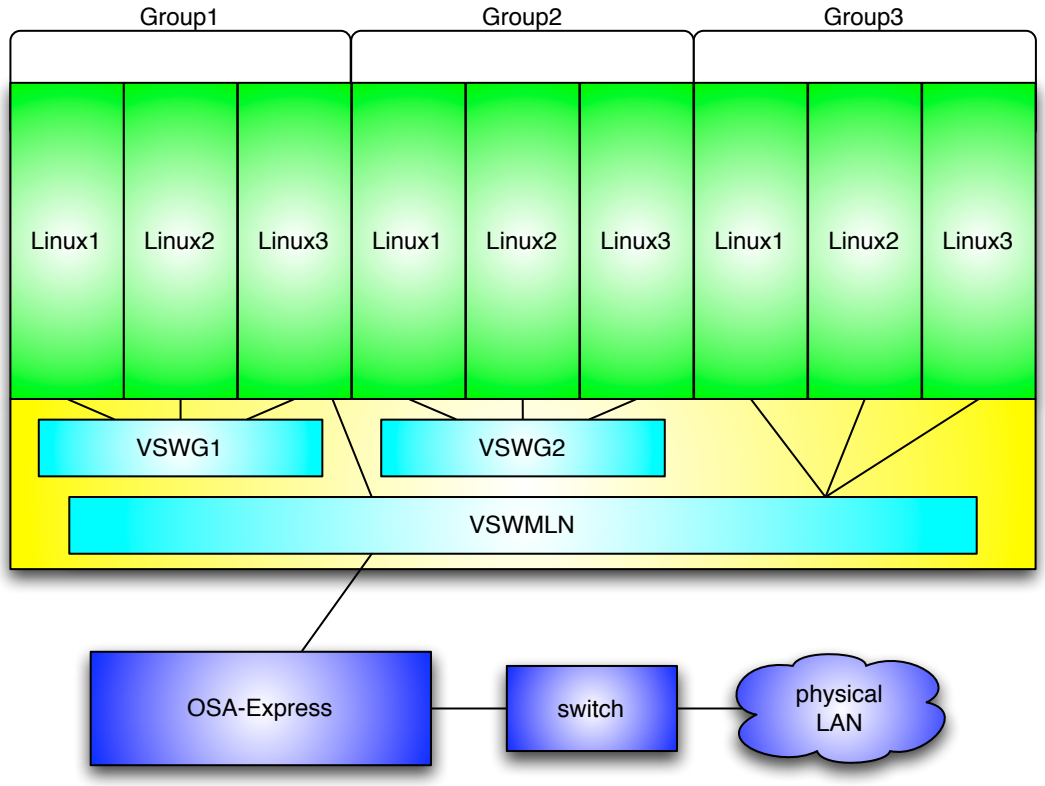

Figure 3.9: *Architecture of an MLN environment using Virtual Switches*

In the case of Group 1, the guests are connected to their own dynam-

ically created VSWITCH VSWG1. In addition, LINUX3 is connected to VSWMLN and can therefor work as a gateway to the physical LAN for the rest of the group.

Group 2 is similar to Group 1, but in this case none of the guests are connected to VSWMLN. Therefor the guests in Group 2 can communicate with each other, but they have no contact beyond their own group.

Group 3 does not have its own VSWITCH, and the guests are therefor connected to VSWMLN by default. This means that they can of course communicate with each other, but it also mean that all of them can reach the physical LAN without going through a guest gateway.

3.6 The scenarios

Three scenarios will be examined to determine how the administration of a z/VM environment has changed with the introduction of the new architecture and MLN. The scenarios will be carried out twice, once on a standard, default z/VM environment and once on the architecture designed in this project and running MLN.

The first scenario will consist of a single guest. The second scenario will consist of six guests and three virtual switches in an arbitrary network topology. The third scenario will consist of eighteen guests and eight virtual switches in an arbitrary network topology.

Chapter 4

Results

The purpose of this chapter is to present the practical implementation of the solutions chosen. Section 4.1 deals with setting up z/MV to receive instructions from MLN, while section 4.4 and section 4.5 deals with MLN and the plugin directly. Section 4.2 describes the storage management solution and section 4.3 describes the network creation and configuration in both Linux and z/VM. Finally, section 4.6 contains the scenarios.

4.1 Programable Operator Facility

PROP is a small and simple service VM. It does not require any special installation or extensive system resources. In fact, most of the already existing guests on the system could probably function as a PROP guest with just a little alteration. The reason for this is that the program files used by PROP resides on MAINTs 0190 disk by default, and practically every normal guest on the system links to this disk. The PROP service VM created for this environment is defined by only three lines in USER DIRECT:

```
1 00001 USER MLNPROP FGRDWAA5 64M 64M ABCDEFG
2 00002    INCLUDE ZEUSCMS
3 00003    MDISK 0191 3390 2850 0005 520W01 MR
```

The five cylinder minidisk is mainly used to store log files and to contain the two PROP specific files PROP RTABLE and PENGUINS EXE.

4.1.1 PROP RTABLE

PROP RTABLE is the routing table used by PROP. The file was written manually for this project as most of its content is tailored to the system created in this project. When PROP receives a message, the first thing it does is look up in the PROP RTABLE file to see what action to take. If it does not find an entry matching the received message, no actions will be taken. This table must be manually created and administrated by the administrator of

the system. Below is a cut-down version of the PROP RTABLE file used in this environment. This is a fully functional PROP RATBLE file, if not a very long one. It will be used here to give an explanation of the attributes and values. The full PROP RTABLE file can be found in Appendix B.

```
1  00000 * * * Top of File * * *
2  00001 LGLOPR MAINT HIOVM2
3  00002 TEXTSYM / $ ^
4  00003 LOGGING ALL
5  00004 ROUTE
6  00005 *
7  00006 *COMMANDS
8  00007 *
9  00008 /XAUTOLOG /               1   9  4 LINUX1         PENGUINS XAUTOLOG
10 00009 /SIGNALSHUTDOWN /         1  15  4 LINUX1         PENGUINS SIGSHUTD
```

Here follows an explanation of lines 2 through 10.

```
LGLOPR MAINT HIOVM2
```

The first line defines the default Logical Operator. The `LGLOPR` attribute begins the statement, while `MAINT` is the name of the guest that is designated as logical operator. This can be any guest that exists on the system, no changes to the guest is necessary. The final part, `HIOVM2`, is the z/VM system ID that the chosen logical operator (`MAINT` in this case) resides on.

```
TEXTSYM / $ ^
```

The `TEXTSYM` statement tells the programmable operator what symbols to interpret as special characters in the text field of the routing table entries. The syntax is as following: `TEXTSYM [blank_sep] [arbchar_sep] [not_symbol]`. If the statement is specified, all three attributes must be entered. A short explanation of the attributes:

`blank_sep`
: This special character indicates that blank characters should be skipped over when scanning the message until is encounters the first non-blank character. This non-blank character is then the beginning of the string that will be compared against the text in the routing table. As an example, the entry /BAR would be triggered by the message BAR but not FOO BAR. FOO BAR fails because there are non-blank characters preceding BAR.

`arbchar_sep`
: This special character indicates that all non matching characters should be skipped over when scanning the message. As an example, the entry $BAR would be triggered by any message containing the string BAR. Both the messages BAR and FOO BAR would in this case trigger the entry.

`not_symbol`
: This special character works as a logical 'not' operator. It is always used with one of the other two special characters. As an example, the entry $^BAR would be triggered by any message that does not contain the string BAR. The message FOO would trigger the entry, while the message FOO BAR would not.

```
LOGGING ALL
```

The `LOGGING` statement tells PROP to what extent it should log information. There are three possible levels. If set to `OFF`, then messages will not be logged. `ON` logs messages, while `ALL` logs messages as well as PROP command responses. There is also an additional optional argument that decides how the strings are to be written to the log file. `UPCASE` makes all the logged text to be written in uppercase while `LOWCASE` logs the text in it original form (`UPCASE` is the default).

```
ROUTE
```

`ROUTE` indicates the end of the configuration section and the beginning of the routing table section.

```
*
*COMMANDS
*
```

The '`*`' is simply the special character indicating the beginning of a comment in the PROP RTABLE file.

```
/XAUTOLOG /               1   9  4 LINUX1         PENGUINS XAUTOLOG
```

This is the first entry in the actual routing table section. An entry in the routing table will always consist of one line (record) and one line only. This is because each part of the entry is assigned a fixed, static number of characters all of which adds up to a total of 72. There are a total of eight fields.

1. The first field contains the comparison text and consists of 25 characters (1-25). In this case the field holds `/XAUTOLOG /`. Because of the '`/`' in front of `XAUTOLOG`, this entry will only be triggered if the first string in the incoming message is 'XAUTOLOG ' (not case sensitive).

2. The second field consists of three characters (27-29), and is the start column of the comparison text. 1 indicates the beginning of the message.

3. The third field consists of three characters (31-33), and is the end column of the comparison text. Since XAUTOLOG with a trailing whitespace consists of nine characters in total, the end column in this example is 9.

4. The fourth field consists of two characters (35-36) and specifies the message class that will trigger the entry. There are a total of nine message classes, and 4 represents messages sent with the command `CP SMSG`. `CM SMSG` messages are special messages in that they are not displayed on the receiving guests console and therefor does not fill up the console buffer. This makes them ideal for sending messages to the guest and not the user of the guest.

5. The fifth field consists of eight characters (38-45) and contains the name of the guest that must be the sender of the message for the entry to be triggered. In this case the entry will only be triggered if the message is sent by LINUX1. If another guest, say LINUX2, should also be able to use the XAUTOLOG command, an additional entry would have to be added to the routing table. This rule would be identical to the one shown here, except LINUX1 would be replaced by LINUX2.

6. The sixth field consists of eight characters (47-54) and contains the z/VM system ID on which the sender resides. In this example, the field is blank meaning that any ID will match.

7. The seventh field consists of eight characters (56-63) and is in this case the name of the exec that will be called when the entry is triggered. `PENGUINS` is an arbitrary REXX script written by the administrator that contains the actual commands that will be executed on the system.

8. The eighth and final field consists of eight characters (65-72) and contains a sting that will be passed to the exec (in this case `PENGUINS`) as a parameter. This is usually to tell the exec what subroutines to run.

The columns 73 and beyond are reserved for future use.

```
/SIGNALSHUTDOWN /         1  15  4 LINUX1         PENGUINS SIGSHUTD
```

This entry was included in the example to show a somewhat special case. First of all, the first part of the actual command consists of two words, SIGNAL SHUTDOWN. However, because PENGUINS expects the target of the command to be the second word in the message, SIGNAL SHUTDOWN was combined to SIGNALSHUTDOWN. This has no effect on the system as it is simply the command that is issued by MLN "behind the sceens". Also notice how the command was shortened to `SIGSHUTD` in the

eighth field because of the eight character limitation. once again this has no effect on the system as this string is only used by PENGUINS.

4.1.2 PENGUINS EXEC

When PROP gets a message from MLN that matches an entry in `PROP RTABLE`, it calls `PENGUINS EXEC` for further actions to be taken. Note that the name of the file, PENGUINS in this case, is not predetermined and can be whatever the administrator wants it to be. This script is written and maintained by the administrator of the z/VM environment and is therefor quite flexible. The scripting language used is REXX. This script is where the actual commands will be sent to the system, and also where some security measures will be implemented. The complete script can be found in Appendix C, but the essential parts will be explained here.

```
00000 * * * Top of File * * *
00001 /* PROP ACTION SCRIPT USED TO VALIDATE COMMANDS AND SO ON */
00002
00003 conffile="RESGUEST CONF"
00004
00005 parse upper arg ruser rnode lglopr msgcode puser pnode netid rtable
00006 pull msg
00007 pull action
00008
00009 /*
00010 * checks if the username is "legal"
00011 */
00012 parse var msg . user restofmsg
00013 okchars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
00014 okchars = okchars !! "0123456789_-$"
00015 if verify(user, okchars, "Nomatch") ! length(user)=0 ! length(user)>8
00016 then do
00017   say "PENGUINS: Syntax error in requested username:" user
00018   exit 1
00019 end
00020
00021 /*
00022 * do not allow a username which is listed in RESGUEST CONF
00023 */
00024 found = 0
00025 parse value stream(conffile,'c','open read') with ok fh
00026 if ok == "ERROR:" then do
00027   say "PENGUINS: Error opening config file" conffile ":" fh
00028   exit 1
00029 end
00030
00031 do while found == 0 & lines(fh) > 0
00032   parse value linein(fh) with cuser .
00033   if translate(cuser) == translate(user) then found = 1
00034 end
00035 ok = stream(fh,'c','close')
00036
00037 if found == 1 then do
00038   say "PENGUINS: Target guest listed as invalid target:" user
00039   exit 1
00040 end
00041
```

```
43 00042 /*
44 00043 * the actuall actions to be taken as a result of the incomming msg
45 00044 */
46 00045 select
47 00046   when action = "XAUTOLOG" then
48 00047     address 'CMS' "XAUTOLOG" user
49 00048   when action = "SIGSHUTD" then
50 00049     address 'CMS' "SIGNAL SHUTDOWN" user
51 00050   otherwise do
52 00051     say "PENGUINS: Unknown action:" action
53 00052     exit 1
54 00053   end
55 00054 end
56 00055 exit 0
```

Most people with some programming background should be able to understand at least the general idea of what the script does. However, a short explanation of the different parts of the script is in order in any case. Note that in the `select` block at line 46, only the commands used in the explanation of PROP RTABLE earlier is included.

```
conffile="RESGUEST CONF"
```

The `RESGUEST CONF` file contains a list of reserved guest names. These are guests that MLN should not be able to influence, typically guests necessary for the continued stable operation of the z/VM environment. Examples being MAINT and OPERATOR.

```
parse upper arg ruser rnode lglopr msgcode puser pnode netid rtable
pull msg
pull action
```

This part parses the incoming parameters to uppercase. It also pulls the the complete message originally sent to PROP and stores it in the variable 'msg' and the parameter from the eighth field of the PROP RTABLE routing table, placing it in the variable 'action'.

```
parse var msg . user restofmsg
okchars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
okchars = okchars !! "0123456789\_-\$"
if verify(user, okchars, "Nomatch") ! length(user)=0 ! length(user)>8
then do
  say "PENGUINS: Syntax error in requested username:" user
  exit 1
end
```

This part extracts the target guest name and checks if it consists of legal characters. It also checks that the name is between one and eight characters in length. If any of these checks fails, the script terminates with an error message.

```
found = 0
parse value stream(conffile,'c','open read') with ok fh
if ok == "ERROR:" then do
  say "PENGUINS: Error opening config file" conffile ":" fh
  exit 1
end

do while found == 0 & lines(fh) > 0
  parse value linein(fh) with cuser .
  if translate(cuser) == translate(user) then found = 1
end
ok = stream(fh,'c','close')

if found == 1 then do
  say "PENGUINS: Target guest listed as invalid target:" user
  exit 1
end
```

Here the file containing the reserved guest names (illegal targets) is opened. Every entry in the file is then compared to the target of the command issued to PROP. If a match is confirmed, the script terminates with an error message. If no match is found, the script continues uninterrupted.

```
select
  when action = "XAUTOLOG" then
    address 'CMS' "XAUTOLOG" user
  when action = "SIGSHUTD" then
    address 'CMS' "SIGNAL SHUTDOWN" user
  otherwise do
    say "PENGUINS: Unknown action:" action
    exit 1
  end
end
```

This is the part of the script where the actual commands to z/VM is chosen. This is based on on the parameter defined in PROP RTABLE. The examples included here corresponds to the routing table entries in the explanation of the PROP RTABLE file. If no match is found, the script terminates with an error message. In the event of an error message at this stage, the chances are strong that the PROP RTABLE and the PENGUINS EXEC files are not synchronized.

The script is critical to the workings of the system in that it must contain every z/VM command requiring higher privilege class than G that MLN should be able to execute. The script can be easily modified to work in similar environments as most of the checks are reasonably general. The part that will most likely require modification is the "select" block since it contains the actual actions the script carries out.

4.2 DirMaint

When setting up DirMaint the main sources of information was the Red-Books

- z/VM Getting Started with Linux on System z9 and zSeries [15]
- Program Directory for IBM z/VM Directory Maintenance Facility Feature [13]

Interaction with DirMaint happens through commands issued by authorized DirMaint administrators. These administrators are defined in the AUTHFOR CONTROL file located on 5VMDIR10. The entries in in this file has the syntax `tUID iUID iNode CmdLevel CmdSets` where:

`tUID`
: `tUID` is the UserID or profileID of the target that access is granted on. This can be one specific ID or the keyword ALL indicating that the commandset can be used on all IDs.

`iUID`
: `iUID` is the UserID that is granted privilege to use the specified commandset.

`iNode`
: `iNode` is the network nodeID that the `iUID` resides on.

`CMSLevel`
: `CMSLevel` is the command level that the authorized user (`iUID`) can submit. This attribute can be either 140A or 150A. 140A allows the user to submit commands using the syntax of DirMaint release 4, while 150A allows the user to utilize the full function of the DirMaint release 5 syntax. The intention of 140A is mainly to be used by programs not yet adapted to use release 5 syntax, but it is usual for a user to be granted both 140A and 150A command level.

`CMSSets`
: `CMSSets` defines the command sets that the authorized user is granted access to. The default IBM command sets are [12]:

  - **A**: Non-DASD user directory Administrator commands.
  - **D**: DASD management user directory administrator commands.
  - **G**: General user commands.
  - **H**: Help Desk commands. Allows looking at things without allowing them to be changed.
  - **M**: Monitoring commands. Allows use of MDAUDIT, PWGEN, PWMON, and SETPW commands.

- **O**: Operational support commands, such as BACKUP, NOTAPE, or SHUTDOWN.
- **P**: Commands needed by automated administration Programs, such as: CLAS, DFSMS, DSO, IPF, NV/AS, RACF.
- **S**: Commands needed by the DirMaint owner and Support programmer.
- **Z**: Commands needed by the DirMaint service machines to communicate with each other.

In the architecture used by MLN, the entire file consists of only four entries:

```
1 ===== * * * Top of File * * *
2 ===== ALL MAINT    *        140A ADGHOPSMZ
3 ===== ALL MAINT    *        150A ADGHOPSMZ
4 ===== ALL MLNPROP  *        140A ADGHOPSMZ
5 ===== ALL MLNPROP  *        150A ADGHOPSMZ
6 ===== * * * End of File * * *
```

As can be seen, MAINT and MLNPROP has permission to act on behalf of all guests with every possible authority level.

As mentioned, authorized users interact with DirMaint by issuing special DirMaint commands. These commands can be easily recognized since the first part is always `DIRMAINT` (or `DIRM` in short form). Most of these commands have a consistent syntax following the pattern
`DIRM [prefix keywords] [command] [arguments]`
A walkthrough will be given of all the DirMaint commands used by MLN. This should make the syntax clear and understandable.

`DIRM ADD LINUX25 LIKE MLNLINUX PW LBYONLY`
The opening part of the command, `DIRM`, tells CMS that this is a DirMaint command and sends it to the DIRMAINT guest. The second part is the command word itself, `ADD`. This tells DirMaint that it should add a new entry in the source directory file called `LINUX25`. `LIKE` tells DirMaint that `LINUX25` should be created using the prototype file `MLNLINUX`. DirMaint will now look through the storage areas it has access to and locate a file called MLNLINUX PROTODIR. In this case, MLNLINUX PROTODIR looks like this

```
1 ===== * * * Top of File * * *
2 ===== USER MLNLINUX NOLOG
3 =====   INCLUDE MLNLINUX
4 ===== * * * End of File * * *
```

DirMaint uses this to add the following entry to the source directory

```
1 ===== * * * Top of File * * *
2 ===== USER LINUX25 LBYONLY
3 =====   INCLUDE MLNLINUX
4 ===== * * * End of File * * *
```

Notice that the name in the prototype file, MLNLINUX, has been replaced with the name specified in the ADD command, LINUX25. DirMaint will only add this entry if it is able to locate the profile MLNLINUX in the source directory file. MLNLINUX is a profile specially customized to run a Linux VM. After the profile has been added manually (DIRM ADD MLNLINUX), it can easily be used by every new Linux guest as it contains all the entries necessary for the guest to operate as a normal machine.

DirMaint also comes with a default profile called LINDFLT specifically tailored for use by Linux guests. LINDFT was however not satisfactory for use in this case as it contains several entries unnecessary or disruptive to the finished guests. It also lacks some entries that will be default for all Linux guests in this system. It was therefor decided to discard LINDFLT and create a completely new profile (MLNLINUX) for use by MLN.

```
00000 * * * Top of File * * *
00001    PROFILE MLNLINUX
00002    CLASS G
00003    DATEFORMAT FULLDATE
00004    IPL CMS PARM AUTOCR
00005    LOGONBY MAINT
00006    MACHINE ESA
00007    OPTION TODENABLE
00008    CONSOLE 0009 3215 T
00009    SPOOL 000C 2540 READER *
00010    SPOOL 000D 2540 PUNCH A
00011    SPOOL 000E 1403 A
00012    LINK MAINT 0190 0190 RR
00013    LINK MAINT 019D 019D RR
00014    LINK MAINT 019E 019E RR
00015    LINK TCPMAINT 0592 0592 RR
00016    LINK MAINT 1000 0191 RR
00017 * * * End of File * * *
```

Most of the entries should be familiar, as they have been explained in earlier examples. A quick description of the unencountered statements follows.

```
CLASS G
```

This defines the privilege class of the new guest. In this case G indicates a normal unprivileged user.

```
LINK MAINT 1000 0191 RR
```

Although the LINK statement has been described before, this entry bears special meaning. MAINTs 1000 mdisk is a small disk of only 10 cylinders containing a single PROFILE EXEC file:

```
00000 * * * Top of File * * *
00001 /* PROFILE EXEC for Linux guests */
00002 'CP SET RUN ON'
00003 'CP SET PF12 RETRIEVE'
00004 'CP SPOOL CONS START TO *'
00005 Say 'IPLing Linux from device 0100'
00006 'CP IPL 100 CLEAR'
00007 /* Should not get here */
00008 Say 'Error IPLing Linux: remaining in CMS'
00009 * * * End of File * * *
```

The special Linux guest specific part of this PROFILE EXEC is the line

```
'CP IPL 100 CLEAR'
```

This entry tells CP to `IPL` the operating system residing on mdisk 0100. Although not yet defined, the new guests 0100 mdisk will be the system disk for the Linux filesystem. In other words, this is the line that will boot Linux.

The last part of the `DIRM ADD` command states `PW LBYONLY`. This sets the z/VM password of the new guest to LBYONLY, as should be the case for every Linux guest.

This concludes the explanation of the `DIRM ADD LINUX25 LIKE MLNLINUX PW LBYONLY` command. Observant readers might have noticed that this command does *not* follow the `DIRM [prefix keywords] [command] [arguments]` syntax mentioned earlier. This is mainly because the command does not target a specific guest, but rather DirMaint itself. The next command however, shows the syntax clearly.

`DIRM FOR LINUX25 STORAGE 64M`
Once again, the command starts with `DIRM`. The next part, `FOR LINUX25`, is the prefix keyword in this command. It tells DirMaint that this command is issued on behalf of LINUX25, in effect targeting the LINUX25 guest. The last part of the command, `STORAGE 64M`, tells DirMaint to set LINUX25s STORAGE attribute to 64M (64 MB).

`DIRM FOR LINUX25 MAXSTORE 64M`
This command differs from the `DIRM FOR LINUX25 STORAGE 64M` command above only in that it sets the MAXSTORAGE attribute instead of the STORAGE attribute.

`DIRM FOR LINUX25 AMDISK 0100 X AUTOG 3330 MLN MR PWS ALL ALL ALL`
This command, like the previous ones, targets LINUX25 (`DIRM FOR LINUX25`). The function of this command is to add a new minidisk, indicated by `AMDISK`, to the target guest. This new mdisk will have the virtual device address

0100 and its type will be determined automatically by DirMaint as indicated by the X. AUTOG 3330 MLN tells DirMaint the size of the mdisk and where this space should be allocated from. 3330 is the size measured in cylinders, while AUTOG and MLN tells DirMaint to automatically allocate the cylinders from the group MLN. This group must be manually defined in the EXTENT CONTROL file on DIRMAINT by a z/VM administrator. Below is a section of the relevant part of the file.

```
1  ===== :REGIONS.
2  =====   *RegionId  VolSer    RegStart      RegEnd  Dev-Type
3  =====    LINUX01   USER03           1         END  3390-03
4  =====    LINUX02   520W01        2855        3234  3390-03
5  =====    LINUX03   520W02        1478        1527  3390-03
6  =====    LINUX04   520W02        1828         END  3390-03
7  ===== :END.
8  ===== :GROUPS.
9  =====   *GroupName RegionList
10 =====  MLN       LINUX01  LINUX02  LINUX03  LINUX04
11 ===== :END.
```

In the :REGIONS. block, all the continuous DASD sections available to DirMaint are listed. Each entry specify the target disk (VolSer), what sort of disk it is (Dev-Type) and the first (RegStart) and last (RegEnd) cylinder of the section. Each entry also has its own unique identifier (RegionId) that is used as a reference in other parts of the file. In the :GROUPS. block, all the groups available to DirMaint are defined. Each group has a unique identifier (GroupName) and a list of regions (RegionList) that makes up the group. In this case, only the beforementioned MLN group is listed, and it consists for the four entries (LINUX01 - LINUX04) defined in the :REGIONS. block.

The next part of the command, MR, sets the default access mode that the guest will get to the mdisk. In this case read-write access will be given unless another user has write or exclusive access to the disk, in which case only read access will be given. The final part of the command, PWS ALL ALL ALL, sets the access passwords (PWS) for *read*, *write* and *multi* access respectfully. These passwords must be supplied by other users to link to the mdisk. In this case, the special value ALL is given, indicating that anyone can link to the disk without providing a password.

DIRM FOR LINUX25 NICDEF 0700 TYPE QDIO LAN SYSTEM VSWITCH5
This is the command used by MLN to define a new Network Interface Card for a guest in z/VM, in this case LINUX25 (FOR LINUX25). NICDEF (Network Interface Card DEFinition) is the command word and the rest of the command describe the new NIC. 0700 is the new cards Virtual Device Address. Each NIC actually uses at least three vdev addresses, so this card will automatically be given 0700, 0701 and 0702. If a guest has multiple NICs then MLN will increment the vdev address with 10, defining the next NIC

with vdev address 0710. Since the NIC will connect to a virtual switch by emulating a QDIO device, the type is set to `TYPE QDIO`. These terms and technologies are more thoroughly described in the *Network* (4.3) section of the *Approach* chapter. `LAN SYSTEM VSWITCH5` tells z/VM that this NIC should be connected to the virtual switch `VSWITCH5`. If VSWITCH5 is an existing virtual switch already defined in z/MV then the NIC will connect to the switch automatically.

`DIRM FOR LINUX25 PURGE NOCLEAN`
The `PRUGE` command is used to completely remove the guest from the system. The target (`LINUX25`) is removed from DirMaints source directory file, any resources it owned are released and all links it had to other resources are dissolved. Of owned resources, mdisks are most noteworthy. The `NOCLEAN` parameter tells DirMaint to release the disk space immediately without erasing its content. This causes the space to become available for automatic (re)allocation right away.

In the original design of the system, the admin linux guest (LINUX1) would issue the `DIRM ADD` commands through PROP. PROP would then go through the normal checks and send the DIRM ADD command to DirMaint. Since PROP has complete authority over DirMaint, the new guest would be added accordingly. LINUX1 would then need authorization to further change and administrate the guest directly. It would therefore send a `DIRM AUTHFOR` command to PROP who would in turn send it to DirMaint. This `DIRM AUTHFOR` command would give LINUX1 complete control over the new guest, allowing DirMaint commands to be issued directly from LINUX1 to DIRMAINT.

The problem with this solution was that the DirMaint commands are normally issued from CMS. When issuing the commands from LINUX1, they go straight to CP which are not able to handle the DirMaint commands directly. Since CMS cannot run simultaneously with Linux on the same guest, a different approach was necessary. The implemented solution was to run all the DirMaint commands through PROP, not just the `DIRM ADD` and `DIRM AUTHFOR` commands. In fact, the `DIRM AUTHFOR` command became unnecessary since LINUX1 would never send a command directly to DirMaint.

4.3 Networking in z/VM and Linux managed by MLN

The implementation of network capability can be divided into three parts.

I) Internal configuration in Linux

II) z/VM configuration for the individual guests

III) Virtual switches in z/VM

The first part, internally configuring Linux, is what would happen on any Linux system. This is namely giving the system an IP address, netmask and broadcast address for each of the used network interfaces. In addition, the network interfaces themselves must be defined. In SLES10 this happens through the creation and manipulation of two files. The IP, netmask and broadcast addressed are defined as attributes in the file /etc/sysconfig/network/ifcfg-qeth-bus-ccw-0.0.xxxx. The "xxxx" is the virtual device address of the NIC that this particular file is for, there are in other words one file for each network interface. As an example, the following file is used by LINUX1s eth0.

`/etc/sysconfig/network/ifcfg-qeth-bus-ccw-0.0.0700`

```
1  BOOTPROTO="static"
2  UNIQUE=""
3  STARTMODE="onboot"
4  IPADDR="10.5.0.17"
5  NETMASK="255.255.255.0"
6  NETWORK="10.5.0.0"
7  BROADCAST="10.5.0.255"
8  _nm_name='qeth-bus-ccw-0.0.0700'
```

The eth0 interface of LINUX1 is connected to a layer 3 virtual switch. Had it been connected to a layer 2 virtual switch (which is the case with guests networked by MLN), an additional attribute would have had to be defined, namely ARP="yes".

The definition of the network interface itself happens in the file /etc/sysconfig/hardware/hwcfg-qeth-bus-ccw-0.0.0.xxxx. Once again, the "xxxx" is the virtual device address of the NIC that this particular file is for. Here is the file for LINUX1s eth0.

`/etc/sysconfig/hardware/hwcfg-qeth-bus-ccw-0.0.0.0700`

```
1  STARTMODE="auto"
2  MODULE="qeth"
3  MODULE_OPTIONS=""
4  MODULE_UNLOAD="yes"
5  SCRIPTUP="hwup-ccw"
6  SCRIPTUP_ccw="hwup-ccw"
7  SCRIPTUP_ccwgroup="hwup-qeth"
8  SCRIPTDOWN="hwdown-ccw"
9  CCW_CHAN_IDS="0.0.0700 0.0.0701 0.0.0702"
10 CCW_CHAN_NUM="3"
11 CCW_CHAN_MODE="OSAP700"
12 QETH_LAYER2_SUPPORT="0"
```

Notice the three distinct channel IDs (CCW_CHAN_IDS). More channel IDs can be defined, but the minimum is three as shown here. Also note that had this NIC been connected to a layer 2 virtual switch rather than a layer 3 virtual switch, QETH_LAYER2_SUPPORT would wave to be set to "1".

Finally, the default gateway is defined in /etc/sysconfig/network/routes

```
1 default 10.5.0.1 - -
```

The second part, z/VM configuration for the individual guest, consists of simply defining the NIC on the z/VM level. As has been explained earlier this is handled by a single entry in the relevant USER DIRECT guest block for each NIC. In the case of LINUX1s NIC, the entry looks like this: `NICDEF 700 TYPE QDIO LAN SYSTEM LOCALNET`. The different parts of this entry as been explained earlier in this text, but attention should be paid to the part `700`. This is the virtual device address of the NIC, and it is vital that it corresponds to the addresses used in the internal Linux configuration part. This is handled automatically by MLN through DirMaint as explained earlier.

The third part, virtual switches in z/VM, mainly consists of generating scripts that creates and destroys Virtual Switches in z/VM. The terms create and destroy is used because a VSWITCH does not have an "off" state. Either it exists and is functioning, or it does not exist. The underlying, statically defined MLN environment VSWITCH is defined in the `SYSTEM CONFIG` file through this line: `DEFINE VSWITCH MLNVSW ETHERNET RDEV 700`. Because it resides in the `SYSTEM CONFIG` file, it is non-volatile and will be recreated every time the system starts. `ETHERNET` tells z/VM that this is a layer 2 switch and `RDEV 700` is the real devise address (`700`) of the physical devised used to communicate with the "outside world". The VSWITCHes that MLN creates as they are needed are created with the CP command `DEFINE VSWITCH` *`name`* `ETHERNET`. Since they are not defined in `SYSTEM CONFIG` they resides completely in memory and is therefor volatile. The VSWITCH has an access control where only specified guests are allowed to connect to the switch. This permission is given through the CP command `SET VSWITCH` *`vswitch`* `GRANT` *`guest`*. In MLN, the VSWITCH is defined and the right permissions given through the automatically generated script start_*VSWITCHname*.sh. This script should be executed before any of the guests using the VSWITCH are started. This is so the guests will find the VSWITCH when they boot and automatically connect to it. If the VSWITCH is started after the guests, the connection will not take place automatically. Note that a guest will boot without problems if it does not find a VSWITCH it is suppose to connect to, but its network connectivity will naturally be affected. Finally, to destroy a VSWITCH the CP command `DETACH VSWITCH` *`name`* is issued. This command is contained in the automatically generated script stop_*VSWITCHname*.sh.

4.4 The MLN Modifications

Since MLN was not originally written to use plugins for the purpose of supporting additional virtualization platforms, some modifications had to be made to its base code. The default virtualization platform in MLN is User-Mode Linux, but it also supports Xen. The code to be executed when using these platforms are hardcoded into MLNs base code, and it often choses what code to run by testing if the "xen" attribute is set for the host.

```
1 if (getScalar("/host/$hostname/xen")){
2   configure_host_XEN($hostname);
3 }
4 else {
5   configure_host_UML($hostname);
6 }
```

As can be seen, it will by default run UML code if the host is not specifically tagged as xen. If the virtualization platform is neither UML or Xen, as is the case in this project, UML code will still be executed. Since it is not desirable to run UML code on a non-UML system, a solution had to be found to run the appropriate plugin code and skip the UML code. The solution was to add strategically placed "hooks" in the base code of MLN. These hooks checks if any plugins contains the routine to be executed, and executes it accordingly if found before MLN proceeds as usual. In cases where UML code has to be skipped over, like the previous example, an exclusive hook is added before the *if* check. In the case of exclusive hooks, a task is considered finished when one code block has carried it out.

Without exclusiveness:

```
1 my $plugin;
2 foreach $plugin (keys %PLUGIN_LIST){
3   my $subcall = $plugin . "_configureSwitch";
4   verbose("calling $subcall\n");
5   if ( defined(&$subcall) ){
6     &$subcall($name);
7   }
8 }
```

With exclusiveness:

```
1 my $each;
2 foreach $each (keys %PLUGIN_LIST){
3   my $call = $each . "_createFilesystem";
4   if ( defined(&$call) ){
5     verbose("Calling plugin method $call\n");
6     return if &$call($hostname);
7   }
8 }
```

The key part of an exclusive hook can be seen on line 6. The return statement makes MLN exit the sub routine without proceeding with the code

after a plugin has successfully executed the task. This addition makes plugins able to "claim" certain operation in the management process. It is the beginning of adding a layer of transparency to MLN where the details are handled by plugins.

4.5 The MLN Plugin

The MLN plugin written in Perl to allow MLN to take advantage of z/VM is attached as Appendix A. While the code is not presented here, a description of the different sub routines can be found in Table 4.1.

Table 4.1: zVM.pl sub routines

Name	**Description**
zVM_postParse	*zVM_postParse* goes through all guests unconditionally and makes general adjustments and changes to suit the environment in which the guests are to be created.
execute	*execute* takes a string as input and either executes it as a command with system() or prints it with out(). It decides what to do by checking the presence of a "dryrun" attribute in the global block of the project. If dryrun is set to "1" then it will print the commands instead of running them, it also wites the scripts to echo their commands instead of running them.
genzname	All guest names in z/VM are restricted to eight characters. *genzname* takes the name of a guest and the project it belongs to as parameters. It then strips away all but the four last characters of each name and combines them to a eight character name used in z/VM. Example: A guest with the name *testguest* belonging to the project *testproject* would get the z/VM name `UESTJECT`. This sub routine is also used for VSWITCHes.

Continues on next page

Name	Description
zVM_createFilesystem	The *zVM_createFilesystem* sub routine does more than its name might imply. It starts off by creating the actual z/VM guest and customizes said guest. A mdisk is added and the appropriate number of network interface cards are defined. It then proceeds with linking and activating the new mdisk and copying the filesystem from a template to the new disk. It also formates the disk for Linux-use if it is necessary. Finally it then unlinks and deactivates the disk.
zVM_mountFilesystem	The *zVM_mountFilesystem* links the mdisk of the target VM to the administrator VM in z/VM. It then activates the disk and mounts it in the administrator Linux VM.
zVM_configure	*zVM_configure* handles the internal configuration of the new filesystem. At this point, that is exclusively network configuration. All the instances of the files /etc/sysconfig/hardware/hwcfg-qeth-bus-ccw-0.0.0.xxxx, /etc/sysconfig/network/ifcfg-qeth-bus-ccw-0.0.xxxx and /etc/sysconfig/network/routes are created in this sub routine.
zVM_unmountFilesystem	The *zVM_unmountFilesystem* unmounts the new mdisk from the administrator Linux VM. It then deactivates the disk and unlinks (detaches) it in z/VM.
zVM_createStartStopScripts	*zVM_createStartStopScripts* creates two scripts for each VM. The start_*bootorder_VMname*.sh script tells z/VM to boot the target guest. The stop_*bootorder_VMname*.sh takes one argument, "kill". If no argument is given, it stops the VM in a clean and orderly fashion. This should be used whenever possible. *kill* simply kills the VM without giving the operating system a chance to shutdown. The *kill* argument should only be used as a last resort when the VM does not respond to other input as it is the equivalent of pulling out the power cord and can potentially damage the filesystem of the VM.

Continues on next page

Name	Description
zVM_checkIfUp	*zVM_checkIfUp* is a simple query sub routine that asks z/VM if the target guest is running. If the guest is running it returns "1", if not it returns "-1".
zVM_configureSwitch	*zVM_configureSwitch* generates two scripts for each virtual switch used to start and stop the switch. The script start_*switchname*.sh dynamically creates the VSWITCH in z/VM. It then grants access to any guest that, according to the MLN project file, should connect to the switch. The stop_*switchname*.sh simply destroys the VSWITCH in z/VM.

4.6 Analyzing the administration complexity through scenarios

Three scenarios will be carried out to examine how the administration of a z/VM environment has changed with the introduction of the new architecture and MLN. The scenarios will be carried out twice, once on a standard, default z/VM environment and once on the architecture design in this project and running MLN. The complexity and size will increase for each scenario:

- Scenario I: 1 guest.
- Scenario II: 6 guests and 3 virtual switches
- Scenario III: 18 guests and 8 virtual switches

The default z/VM environment will also have a LINUX1 administrator Linux guest, and when the scenarios begins the mdisks containing the templates will already be linked to LINUX1. These templates will contain the appropriate general network settings of the network, like broadcast address and netmask. As an example, all templates used in scenario III will be configured for network 10.0.0.0 with netmask 255.255.255.0. Also, both environments will have an already defined outgoing virtual switch named GATEWAY in z/VM. The scenarios will have two main parts. Part one will be the creation of the architecture described, while part two will consist of administration tasks on the created environment.

There will basically two tasks considered in part two.
Administration tasks:

Startup
: Starting a single VM, starting half of the VMs and starting the entire site.

Check if up
Check if a single VM is up, check if half of the VMs are up and check if the entire site is up. Only the status of the specified VMs will be checked here. MLN can for instance not check the entire project when checking half of the VMs.

Shutdown
Shutdown was considered as a task, but it was decided not to include it. The reason being that the task is identical to startup except for the number of characters in the command used. (Example `SINGNAL SHUTDOWN` instead of `XAUTOLOG` for Non-MLN approach)

Restart
Restarting a VM was also considered, but as this would simply be Startup + Shutdown for both MLN and Non-MLN approach.

Metrics that will be considered in both creation and administration are:

Number of commands issued
The number of commands needed to be issued can be seen as an indication of the complexity of the task and the amount of time needed to complete it. This can of course be an extremely crude assessment as the complexity of the commands themselves vary greatly, but it is one of the metrics that will make up the evaluation as a whole.

Number of characters used on commands
The total number of characters used to make up the commands issued will be used to supplement the "number of commands issued" metrics. This is because the length of the command can give an indication of the commands complexity. As an example "QUERY NAMES" and "DIRM FOR LINUX25 AMDISK 0100 X AUTOG 3338 MLN MR PWS ALL ALL ALL" will both be noted as one command, but the number of characters will show that they are clearly not equal.

Metrics that will only be considered in creation are:

Number of lines written to file
The number of lines written to file is a count of all lines written to any files. As with commands issued, this can be seen as an indication of a tasks complexity and time consumption. If a line is copied and modified, it will count as written. Blank lines inserted to make the files easier to read will not be counted.

Number of characters written to file
As with "number of characters used on commands", number of characters written to file will be used to supplement "numbers of lines written to file". Again this is to indicate the difference in complexity.

In several scripting languages, normal human written code spanning several pages can be reformatted to one really long line. Tricks like that will of course not be used in the scenarios, but it is important to note that one metric alone proves to unreliable. Copied lines will not affect this metric.

Number of lines copied
The number of lines copied in files is taken into account because of the small amount of work and time it takes to copy a line. An action will be counted in this category only if the line is copied without being modified afterwards. Also, the line must first have been written once during the scenario before it is copied.

Number of files edited
The number of files edited gives an indication of the complexity and level of consolidation of information in the administration process.

Number of systems logged on to
The number of systems logged on to tells us how many systems had to be logged on to (not just accessed). This gives an indication to the centralization of the administration process.

4.6.1 Scenario I

In this first scenario, a single web server guest will be created and connected to the existing outgoing gateway switch as shown in Figure 4.1. The template for the web server will be /dev/dasdb on LINUX1. The attributes unique to each host, like IP address, must be set accordingly.

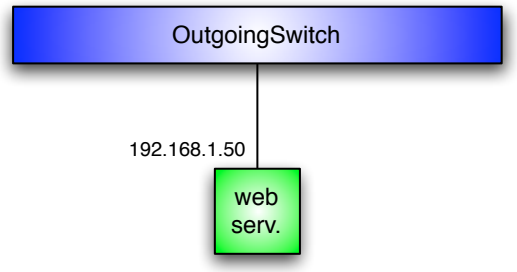

Figure 4.1: *Scenario I Architecture Overview*

The non-MLN approach

Create

First z/VM is logged on to as the user MAINT, then the creation of the sites z/VM layer can begin. The new web server guest must be granted access to the outgoing gateway switch.

```
SET VSWITCH GATEWAY GRANT WS1RIO1
```

Then the guest itself is created by defining it in the USER DIRECT file
`X USER DIRECT`

```
1  USER WS1RIO1 1G 1G G
2          DATEFORMAT FULLDATE
3          IPL CMS PARM AUTOCR
4          LOGONBY MAINT
5          MACHINE ESA
6          OPTION TODENABLE
7          CONSOLE 0009 3215 T
8          SPOOL 000C 2540 READER *
9          SPOOL 000D 2540 PUNCH A
10         SPOOL 000E 1403 A
11         NICDEF 700 TYPE QDIO LAN SYSTEM GATEWAY
12         LINK MAINT 0190 0190 RR
13         LINK MAINT 019D 019D RR
14         LINK MAINT 019E 019E RR
15         LINK TCPMAINT 0592 0592 RR
16         LINK MAINT 1000 0191 RR
17         MDISK 0100 3390 0001 24000 DISK01 MR ALL ALL ALL
```

After the guest have been defined, the entry for MAINT is located and the following line added

```
MDISK 1000 3390 2840 010 520W01  MR READ     WRITE    MULTIPLE
```

When all changes have been made to USER DIRECT the file is saved and exited with the command

```
FILE
```

A check is executed to make sure USER DIRECT has no syntax error

```
DIRECTXA USER DIRECT (EDIT
```

and if the no syntax errors were found, USER DIRECT is re-read to the system

```
DIRECTXA USER DIRECT
```

Now MAINT must access its new disk and create the appropriate PROFILE EXEC file

```
ACCESS 1000 X
X PROFILE EXEC X
```

```
1 /* PROFILE EXEC for Linux guests */
2 'CP SET RUN ON'
3 'CP SET PF12 RETRIEVE'
4 'CP SPOOL CONS START TO *'
5 Say 'IPLing Linux from device 0100'
6 'CP IPL 100 CLEAR'
7 Say 'Error IPLing Linux: remaining in CMS'
```

```
FILE
RELEASE X
```

The part of the creation process that takes place in z/VM is now complete. The next part takes place on LINUX1. After logging on to LINUX1 it is time to copy the template on to the new guest and configure its network settings.

First the mdisk of the guest is linked to LINUX1, placed online and then formated. The linked disk will appear as /dev/dasdc since this is the next available designation.

```
vmcp link ws1rio1 0100 8472 mr
chccwdev -e 8472
dasdfmt -b 4096 -y -f /dev/dasdc
```

The appropriate template, in this case /dev/dasdb which is the web server template, is then applied to the newly formated disk

```
dd if=/dev/dasdb of=/dev/dasdc
chccwdev -d 8472
chccwdev -e 8472
```

The disk is then mounted so the filesystem can be manipulated

```
mount /dev/dasdc1 /mnt/mdisk
```

Now the guest must be configured with the correct network settings

```
cd /mnt/mdisk/etc/sysconfig/network/
```

`joe ifcfg-qeth-bus-ccw-0.0.0.0700`
The IP address is set to 192.168.1.50

```
IPADDR=192.168.1.50
```

Finally the hostname is changed
`joe /mnt/mdisk/etc/HOSTNAME`

```
WebServer1
```

Now that all changes are complete, the disk can be unmounted, placed offline and detached
`umount /mnt/mdisk`
`chccwdev -d 8472`
`vmcp det 8472`
The new guest is now ready to be used

Table 4.2: Scenario I Create: The non-MLN approach

Number of commands issued	22
Number of characters used on commands	448
Number of lines written to file	24
Number of characters written to file	550
Number of lines copied	0
Number of files edited	4
Number of systems logged on to	2

Administrate

Since this scenario only contains one VM, the "half of the VMs " and "entire site" will be one. In the non-MLN approach, the administrative tasks takes place in z/VM.

One VM

Startup:

`XAUTOLOG WS1RIO1`

Check if up:

`Q USERS WS1RIO1`

Table 4.3: Scenario I Administrate: The non-MLN approach

	One	Half	All
Startup			
Number of commands issued	1	1	1
Number of characters used on commands	16	16	16
Check if up			
Number of commands issued	1	1	1
Number of characters used on commands	15	15	15

The MLN approach

Create

LINUX1 is logged on to and the entire creation process takes place form this system. When creating the site with MLN, the whole architecture is described in the project file.

First the project file is created
`joe scenario1.mln`

```
global {
  project scenario1
  $netmask 255.255.255.0
  $broadcast_address 192.168.1.255
}

host WS1 {
  template /dev/dasdc
  size 20G
  memory 1G
  network eth0 {
    netmask $netmask
    broadcast $broadcast_address
    address 192.168.1.50
  }
}
```

Then the project is built
`mln build -f scenario1.mln`
The new web server guest is now ready to be used

Administrate

Since this scenario only contains one VM, the "half of the VMs " and "entire site" will be one. In the MLN approach, the administrative tasks takes place in Linux

Table 4.4: Scenario I Create: The MLN approach

Number of commands issued	2
Number of characters used on commands	43
Number of lines written to file	15
Number of characters written to file	208
Number of lines copied	0
Number of files edited	1
Number of systems logged on to	1

One VM

Startup:

```
mln start -p scenario1 -h sw1
```

Check if up:

```
mln status -p scenario1 -h sw1
```

Entire site

Startup:

```
mln start -p scenario1
```

Check if up:

```
mln status -p scenario1
```

Table 4.5: Scenario I Administrate: The MLN approach

	One	**Half**	**All**
Startup			
Number of commands issued	1	1	1
Number of characters used on commands	29	29	22
Check if up			
Number of commands issued	1	1	1
Number of characters used on commands	30	30	23

4.6.2 Scenario II

In the second scenario, an environment with three independent but identical web servers will be created. Each web server will have a gateway as a link to the physical network as shown in Figure 4.2. The reason virtual switches are used instead of point-to-point connections is partially because MLN does not support point-to-point and partially because switches makes the architecture more open to changes in the future.

LINUX1 will have the following templates:

- /dev/dasdb -> gateway
- /dev/dasdc -> web server

Tier 1
Tier 1 consists of three VMs working as gateways. Each gateway stands in front of their own independent but identical virtual LAN.

Tier 2
Tier 2 consists of three VMs running web servers. These are completely independent of each other, and can therefore have completely identical configuration in the Linux layer. Their z/VM configuration will of course not be identical.

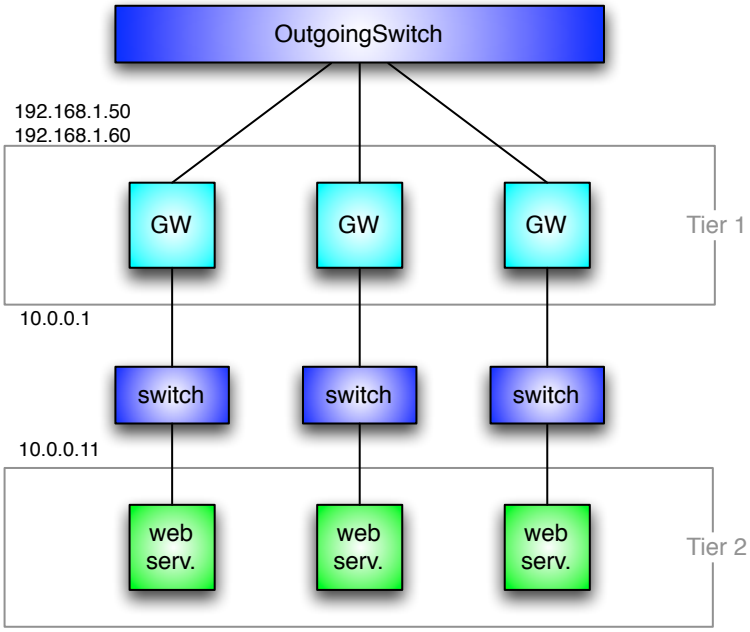

Figure 4.2: *Scenario II Architecture Overview*

The non-MLN approach

Create

First z/VM is logged on to as the user MAINT, then the creation of the sites z/VM layer can begin. The first part to be created are the VSWITCHes, along with granting guests access to them.

```
DEFINE VSWITCH SW1RIO2 ETHERNET
DEFINE VSWITCH SW2RIO2 ETHERNET
DEFINE VSWITCH SW3RIO2 ETHERNET
SET VSWITCH GATEWAY GRANT GW1RIO2
SET VSWITCH GATEWAY GRANT GW2RIO2
SET VSWITCH GATEWAY GRANT GW3RIO2
SET VSWITCH SW1RIO2 GRANT GW1RIO2
SET VSWITCH SW2RIO2 GRANT GW2RIO2
SET VSWITCH SW2RIO2 GRANT GW3RIO2
SET VSWITCH SW1RIO2 GRANT WS1RIO2
SET VSWITCH SW2RIO2 GRANT WS2RIO2
SET VSWITCH SW2RIO2 GRANT WS3RIO2
```

Then the guests themselves are created by defining them in the USER DIRECT file

```
X USER DIRECT
```

```
1  PROFILE MLNLINUX
2          CLASS G
3          DATEFORMAT FULLDATE
4          IPL CMS PARM AUTOCR
5          LOGONBY MAINT
6          MACHINE ESA
7          OPTION TODENABLE
8          CONSOLE 0009 3215 T
9          SPOOL 000C 2540 READER *
10         SPOOL 000D 2540 PUNCH A
11         SPOOL 000E 1403 A
12         LINK MAINT 0190 0190 RR
13         LINK MAINT 019D 019D RR
14         LINK MAINT 019E 019E RR
15         LINK TCPMAINT 0592 0592 RR
16         LINK MAINT 1000 0191 RR
17
18 USER GW1RIO2 1G 1G
19         PROFILE MLNLINUX
20         NICDEF 700 TYPE QDIO LAN SYSTEM GATEWAY
21         NICDEF 710 TYPE QDIO LAN SYSTEM SW1RIO2
22         MDISK 0100 3390 0001 1665 DISK01 MR ALL ALL ALL
23 USER GW1RIO2 1G 1G
24         PROFILE MLNLINUX
25         NICDEF 700 TYPE QDIO LAN SYSTEM GATEWAY
26         NICDEF 710 TYPE QDIO LAN SYSTEM SW2RIO2
27         MDISK 0100 3390 1666 3330 DISK01 MR ALL ALL ALL
28 USER GW1RIO2 1G 1G
29         PROFILE MLNLINUX
30         NICDEF 700 TYPE QDIO LAN SYSTEM GATEWAY
```

```
        NICDEF 710 TYPE QDIO LAN SYSTEM SW3RIO2
        MDISK 0100 3390 0001 1665 DISK02 MR ALL ALL ALL

USER WS1RIO2 1G 1G
        PROFILE MLNLINUX
        NICDEF 700 TYPE QDIO LAN SYSTEM SW1RIO2
        MDISK 0100 3390 0001 24000 DISK03 MR ALL ALL ALL
USER WS2RIO2 1G 1G
        PROFILE MLNLINUX
        NICDEF 700 TYPE QDIO LAN SYSTEM SW2RIO2
        MDISK 0100 3390 0001 24000 DISK04 MR ALL ALL ALL
USER WS3RIO2 1G 1G
        PROFILE MLNLINUX
        NICDEF 700 TYPE QDIO LAN SYSTEM SW3RIO2
        MDISK 0100 3390 0001 24000 DISK05 MR ALL ALL ALL
```

After these guests have been defined, the entry for MAINT is located and the following line added

```
MDISK 1000 3390 2840 010 520W01  MR READ     WRITE    MULTIPLE
```

When all changes have been made to USER DIRECT the file is saved and exited with the command

```
FILE
```

A check is executed to make sure USER DIRECT has no syntax error

```
DIRECTXA USER DIRECT (EDIT
```

and if the no syntax errors were found, USER DIRECT is re-read to the system

```
DIRECTXA USER DIRECT
```

Now MAINT must access its new disk and create the appropriate PROFILE EXEC file

```
ACCESS 1000 X
X PROFILE EXEC X
```

```
/* PROFILE EXEC for Linux guests */
'CP SET RUN ON'
'CP SET PF12 RETRIEVE'
'CP SPOOL CONS START TO *'
Say 'IPLing Linux from device 0100'
'CP IPL 100 CLEAR'
Say 'Error IPLing Linux: remaining in CMS'
```

```
FILE
RELEASE X
```

The part of the creation process that takes place in z/VM is now complete. The next part takes place on LINUX1. After logging on to LINUX1 it is time to copy the templates on to the new guests and configure their network settings. This process is basically identical for all the new guests and so will not be shown for all 6. The example shown here is of one of the gateways.

First the mdisk of the guest is linked to LINUX1, placed online and then formated. The linked disk will appear as /dev/dasdd since this is the next

available designation.

```
vmcp link gw1rio2 0100 8472 mr
chccwdev -e 8472
dasdfmt -b 4096 -y -f /dev/dasdd
```

The appropriate template, in this case /dev/dasdb which is the SQL server template, is then applied to the newly formated disk

```
dd if=/dev/dasdb of=/dev/dasdd
chccwdev -d 8472
chccwdev -e 8472
```

The disk is then mounted so the filesystem can be manipulated

```
mount /dev/dasdd1 /mnt/mdisk
```

Now the guest must be configured for its place in the network

```
cd /mnt/mdisk/etc/sysconfig/network/
```

Because the gateway has two network interface cards, one must be added manually

```
cp ifcfg-qeth-bus-ccw-0.0.0.0700 ifcfg-qeth-bus-ccw-0.0.0.0710
joe ifcfg-qeth-bus-ccw-0.0.0.0700
```

This is the NIC that connects the gateway to the outgoing gateway switch. The IP address is set to 192.168.1.50 and the broadcast address is set to 192.168.1.255

```
IPADDR=192.168.1.50
BROADCAST=192.168.1.255
```

The outgoing NIC also needs a gateway on the outside network

```
joe routes
```

The following line is entered

```
default 192.168.1.42 - -
```

```
joe ifcfg-qeth-bus-ccw-0.0.0.0710
```

This is the NIC used by the gateway to connect to the private LAN. Since the three LANs are separated form each other, all the gateways will use the local address 10.0.0.1, and all the web servers will use 10.0.0.5

```
IPADDR=10.0.0.1
_nm_name='qeth-bus-ccw-0.0.0710'
```

The hardware definition must also be updated for the new card

```
cd /mnt/mdisk/etc/sysconfig/hardware/
cp hwcfg-qeth-bus-ccw-0.0.0.0700 hwcfg-qeth-bus-ccw-0.0.0.0710
joe hwcfg-qeth-bus-ccw-0.0.0.0710
```

```
CCW_CHAN_IDS="0.0.0710 0.0.0711 0.0.0712"
```

Finally the hostname is changed

```
joe /mnt/mdisk/etc/HOSTNAME
```

```
Gateway1
```

Now that all changes are complete, the disk can be unmounted, placed offline and detached

```
umount /mnt/mdisk
chccwdev -d 8472
vmcp det 8472
```

The new guest is now ready to be used

The web servers only has one NIC, and so the creation and configuration of the second NIC in the gateways will of course not take place in the web servers.

Table 4.6: Scenario II Create: The non-MLN approach

Number of commands issued	116
Number of characters used on commands	3066
Number of lines written to file	67
Number of characters written to file	1680
Number of lines copied	7
Number of files edited	23
Number of systems logged on to	2

Administrate

Since this scenario contains six VMs, the "half of the VMs " will consist of the three web servers. In the non-MLN approach, the administrative tasks takes place in z/VM.

One VM

Startup:

```
XAUTOLOG WS1RIO2
```

Check if up:

```
Q USERS WS1RIO2
```

Half of the VMs

Startup:

```
XAUTOLOG WS1RIO2
XAUTOLOG WS2RIO2
XAUTOLOG WS3RIO2
```

Check if up:

```
Q USERS WS1RIO2
Q USERS WS2RIO2
Q USERS WS3RIO2
```

Entire site

Startup:

```
XAUTOLOG WS1RIO2
XAUTOLOG WS2RIO2
XAUTOLOG WS3RIO2
XAUTOLOG GW1RIO2
XAUTOLOG GW2RIO2
XAUTOLOG GW3RIO2
```

Check if up:

```
Q USERS WS1RIO2
Q USERS WS2RIO2
Q USERS WS3RIO2
Q USERS GW1RIO2
Q USERS GW2RIO2
Q USERS GW3RIO2
```

Table 4.7: Scenario II Administrate: The non-MLN approach

	One	**Half**	**All**
Startup			
Number of commands issued	1	3	6
Number of characters used on commands	16	48	96
Check if up			
Number of commands issued	1	3	6
Number of characters used on commands	15	45	90

The MLN approach

Create

LINUX1 is logged on to and the entire creation process takes place form this system. When creating the site with MLN, the whole architecture is described in the project file. Note that because each of the web servers and gateways connects to different switches, autoenum cannot be used.

First the project file is created

```
joe scenario2.mln
```

```
global {
  project scenario2
  $LAN_broadcast = 10.0.0.255
  $GW_broadcast = 192.168.1.255
  $GW_gateway = 192.168.1.42
  $netmask = 255.255.255.0
  $WS_IP_address = 10.0.0.5
  $GW_IP_address = 10.0.0.1
}

superclass gateway {
  template /dev/dasdb
  size 1G
  memory 1G
  network eth0 {
        netmask $netmask
        broadcast $GW_broadcast
        gateway $GW_gateway
  }
  network eth1 {
    netmask $netmask
    broadcast $LAN_broadcast
    address $GW_IP_address
  }
}

superclass webserver {
  template /dev/dasdc
  size 20G
  memory 1G
  network eth0 {
    netmask $netmask
    broadcast $LAN_broadcast
    address $WS_IP_address
  }
}

host GW1 {
  superclass gateway
  network eth0 {
    address 192.168.1.51
  }
  network eth1 {
    switch vsw1
  }
}

host GW2 {
  superclass gateway
  network eth0 {
    address 192.168.1.52
  }
  network eth1 {
    switch vsw2
  }
}

host GW3 {
  superclass gateway
  network eth0 {
    address 192.168.1.53
  }
  network eth1 {
    switch vsw3
  }
}

host WS1 {
  superclass webserver
  network eth0 {
    switch vsw1
  }
}

host WS2 {
  superclass webserver
  network eth0 {
    switch vsw2
  }
}

host WS3 {
  superclass webserver
  network eth0 {
        switch vsw3
  }
}

switch vsw1 { }
switch vsw2 { }
switch vsw3 { }
```

Then the project is built

```
mln build -f scenario2.mln
```

The site is now ready to be used

Administrate

Since this scenario contains six VMs, the "half of the VMs " will consist of the three web servers. In the MLN approach, the administrative tasks takes place in Linux.

Table 4.8: Scenario II Create: The MLN approach

Number of commands issued	2
Number of characters used on commands	43
Number of lines written to file	48
Number of characters written to file	736
Number of lines copied	34
Number of files edited	1
Number of systems logged on to	1

One VM

Startup:

```
mln start -p scenario2 -h ws1
```

Check if up:

```
mln status -p scenario2 -h ws1
```

Half of the VMs

Startup:

```
mln start -p scenario2 -h ws1
mln start -p scenario2 -h ws2
mln start -p scenario2 -h ws3
```

Check if up:

```
mln status -p scenario2 -h ws1
mln status -p scenario2 -h ws2
mln status -p scenario2 -h ws3
```

Entire site

Startup:

```
mln start -p scenario2
```

Check if up:

```
mln status -p scenario2
```

Table 4.9: Scenario II Administrate: The MLN approach

	One	**Half**	**All**
Startup			
Number of commands issued	1	3	1
Number of characters used on commands	29	87	22
Check if up			
Number of commands issued	1	3	1
Number of characters used on commands	30	90	23

4.6.3 Scenario III

The scenario will revolve around a site used for web serving. The scenario architecture [Figure 4.3] consists of four tiers of virtual machines. The tiers will be connected with each others through layers of virtual switches.

LINUX1 will have the following templates:

- /dev/dasdb -> load balancer
- /dev/dasdc -> web server
- /dev/dasdd -> SQL server

Tier 1

Tier 1 consists of four VMs tasked with load balancing the incoming traffic. Since this is the first tier in the architecture, failure in this part would cause the entire system to become unavailable to anyone on the outside. Because of this, four VMs are used for the sake of redundancy.

Tier 2

Tier 2 consists of eight VMs running the main service of the site, the web servers. Because of this, a relatively high number of VMs resides in this tier. Both to provide a high level of redundancy, and to spread the load over several VMs.

Tier 3

Tier 3 consists of two VMs tasked with load balancing the traffic between the web servers in tier 2 and the SQL servers in tier 4. Each VM serves four web servers and links to all the SQL servers on tier 4.

Tier 4

Tier 4 consists of four VMs running SQL servers for the web servers in tier 2. Each server can be accessed by any of the web servers. This is to ensure redundancy and to make it possible to balance the load as equally as possible over each of the servers.

The non-MLN approach

Create

First z/VM is logged on to as the user MAINT, then the creation of the sites z/VM layer can begin. The first part to be created are the VSWITCHes, along with granting guests access to them.
Because of the number of commands needed to create and configure the VSWITCHes, they have been omitted from this report but can be found in

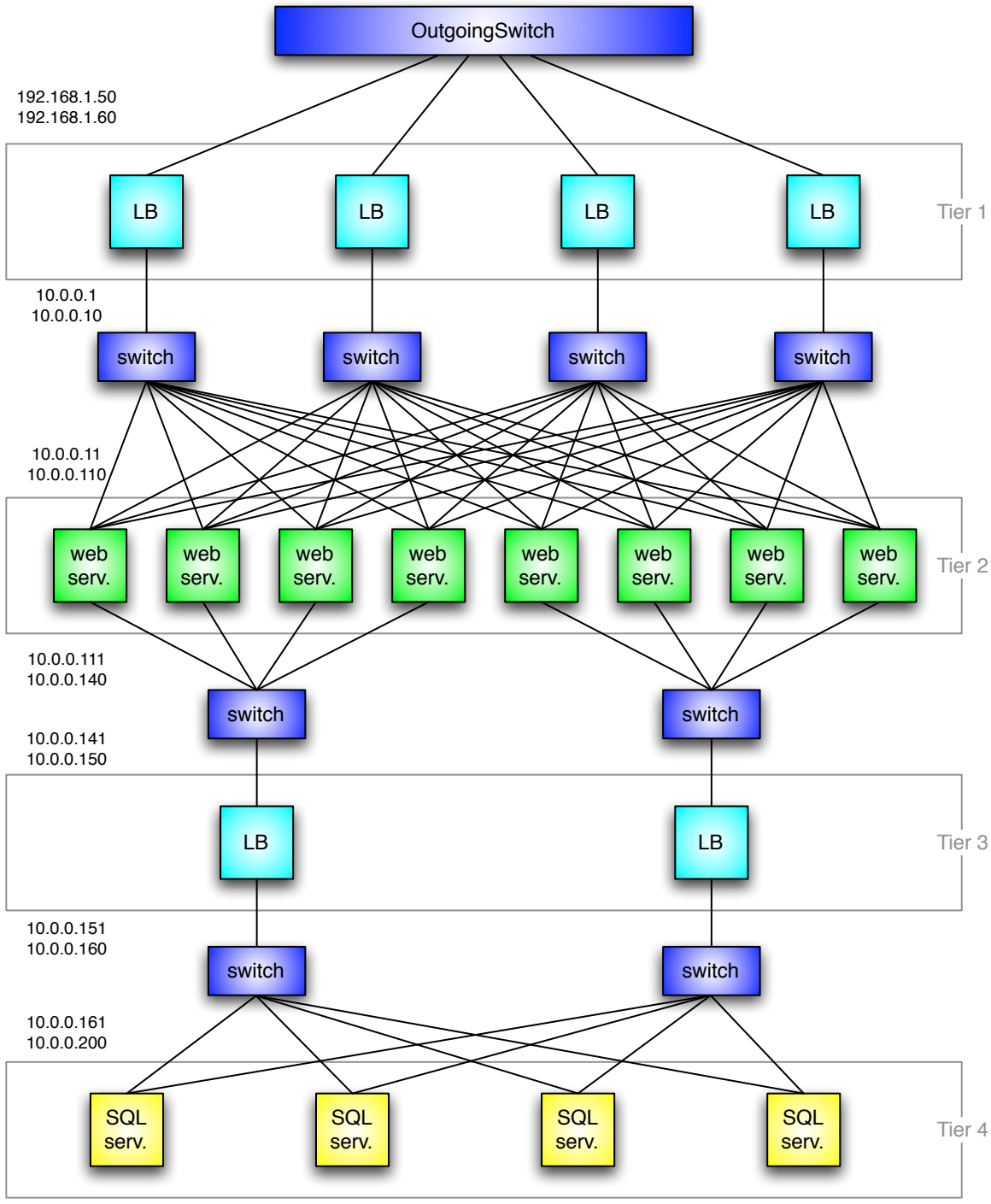

Figure 4.3: *Scenario III Architecture Overview*

Appendix G
Then the guests themselves are created by defining them in the USER DIRECT file
`X USER DIRECT`
Because of the size of the USER DIRECT section, it can be found appended to this report as Appendix H
After these guests have been defined, the entry for MAINT is located and the following line added

```
MDISK 1000 3390 2840 010 520W01  MR READ     WRITE    MULTIPLE
```

When all changes have been made to USER DIRECT the file is saved and exited with the command
`FILE`
A check is executed to make sure USER DIRECT has no syntax error
`DIRECTXA USER DIRECT (EDIT`
and if the no syntax errors were found, USER DIRECT is re-read to the system
`DIRECTXA USER DIRECT`
Now MAINT must access its new disk and create the appropriate PROFILE EXEC file
`ACCESS 1000 X`
`X PROFILE EXEC X`

```
1 /* PROFILE EXEC for Linux guests */
2 'CP SET RUN ON'
3 'CP SET PF12 RETRIEVE'
4 'CP SPOOL CONS START TO *'
5 Say 'IPLing Linux from device 0100'
6 'CP IPL 100 CLEAR'
7 Say 'Error IPLing Linux: remaining in CMS'
```

`FILE`
`RELEASE X`
The part of the creation process that takes place in z/VM is now complete. The next part takes place on LINUX1. After logging on to LINUX1 it is time to copy the templates on to the new guests and configure their network settings. This process is basically identical for all the new guests and so will not be shown for all 18. The example shown here is of one of the SQL servers.

First the mdisk of the guest is linked to LINUX1, placed online and then formated. The linked disk will appear as /dev/dasde since this is the next available designation.
`vmcp link qls1rio3 0100 8472 mr`
`chccwdev -e 8472`
`dasdfmt -b 4096 -y -f /dev/dasde`
The appropriate template, in this case /dev/dasdd which is the SQL server

template, is then applied to the newly formated disk

```
dd if=/dev/dasdd of=/dev/dasde
chccwdev -d 8472
chccwdev -e 8472
```

The disk is then mounted so the filesystem can be manipulated

```
mount /dev/dasde1 /mnt/mdisk
```

Now the guest must be configured for its place in the network

```
cd /mnt/mdisk/etc/sysconfig/network/
joe ifcfg-qeth-bus-ccw-0.0.0.0700
```

The IP address is set to 10.0.0.161

```
IPADDR=10.0.0.161
```

Because the SQL server has two network interface cards, one must be added manually

```
cp ifcfg-qeth-bus-ccw-0.0.0.0700 ifcfg-qeth-bus-ccw-0.0.0.0710
joe ifcfg-qeth-bus-ccw-0.0.0.0710
```

Some changes are necessary

```
IPADDR=10.0.0.162
_nm_name='qeth-bus-ccw-0.0.0710'
```

The hardware definition must also be updated for the new card

```
cd /mnt/mdisk/etc/sysconfig/hardware/
cp hwcfg-qeth-bus-ccw-0.0.0.0700 hwcfg-qeth-bus-ccw-0.0.0.0710
joe hwcfg-qeth-bus-ccw-0.0.0.0710
```

```
CCW_CHAN_IDS="0.0.0710 0.0.0711 0.0.0712"
```

Finally the hostname is changed

```
joe /mnt/mdisk/etc/HOSTNAME
```

```
SQLserver1
```

Now that all changes are complete, the disk can be unmounted, placed offline and detached

```
umount /mnt/mdisk
chccwdev -d 8472
vmcp det 8472
```

The new guest is now ready to be used

From this example it is clear that if the guest has more than one NIC, the number of commands increase with 1+4*number of NICs beyond the first one. Since all guests in this scenario have more than one NIC, the total number of commands used per guest while in LINUX1 is 14+4*(NIC-1). Also the number of files edited increases with 2 per NIC and the number of lines written to file increases with 3 per NIC. In addition to this each of the load balancers in tier 1 has one more line written to file since the broadcast address of their outgoing NIC must be changed as well.

In the end, this is how the metrics were calculated:
Number of commands issued
76 commands in z/VM
10 guests with 2 NICs: 10*(14+4*(2-1))+4[1]=184
8 guests with 5 NICs: 8*(14+4*(5-1))=240
Number of characters used on commands
2405 from z/VM
10 guests with 2 NICs: 10*(347+190*(2-1))+4[2]+(4*10)[3]=5414
8 guests with 5 NICs: 8*(347+190*(5-1))=8856
Number of lines written to file
75 lines in z/VM 10 guests with 2 NICs: 10*(2+3*(2-1))+4[4]+4[5]=58
8 guests with 5 NICs: 8*(2+3*(5-1))=112
Number of characters written to file
2348 characters from z/VM
10 guests with 2 NICs: 10*(15+88*(2-1))+(3*6)[6]+4[7]+(12*2)[8]+(4*4)[9]+(23*4)[10]+(4*24)[11]=1376
8 guests with 5 NICs: 8*(15+88*(5-1))+32[12]+(8*2)[13]=3064
Number of lines copied
60 lines in z/VM
Number of files edited
2 files in z/VM
10 guests with 2 NICs: 10*(2+2*(2-1))+4[14]=44
8 guests with 5 NICs: 8*(2+2*(5-1))=80

[1]Four guests must change gateway
[2]The name of the SQL servers are one character longer than the other guests
[3]The entries for the outside gateway is 10 characters long
[4]The load balancers in tier 1 must in addition change the broadcast address of their outgoing NIC
[5]The entry for a new gateway is 1 line
[6]The load balancers uses three more characters in their name
[7]Four NICs have one extra number in the IP address
[8]12 NICs have two extra digits in their IP address
[9]Four NICs are connected to the external network. These have four more digits in their IP address
[10]The four NICs connected to the external network must change broadcast address. Each entry takes 23 characters
[11]The entry for the gateway to the outside network is 24 characters long
[12]32 NICs have one extra digit in the IP address
[13]8 NICs have two extra digits in their IP address
[14]One file for each gateway changed

Table 4.10: Scenario III Create: The non-MLN approach

Number of commands issued	500
Number of characters used on commands	16675
Number of lines written to file	245
Number of characters written to file	6788
Number of lines copied	60
Number of files edited	126
Number of systems logged on to	2

Administrate

Since this scenario contains eighteen VMs, the "half of the VMs " will consist of the nine VMs on the left half of Figure 4.3. In the non-MLN approach, the administrative tasks takes place in z/VM.

One VM

Startup:

```
XAUTOLOG WS1RIO3
```

Check if up:

```
Q USERS WS1RIO3
```

Half of the VMs

Startup:

```
XAUTOLOG LB1RIO3
XAUTOLOG LB2RIO3
XAUTOLOG WS1RIO3
XAUTOLOG WS2RIO3
XAUTOLOG WS3RIO3
XAUTOLOG WS4RIO3
XAUTOLOG LB5RIO3
XAUTOLOG QLS1RIO3
XAUTOLOG QLS2RIO3
```

Check if up:

```
Q USERS LB1RIO3
Q USERS LB2RIO3
Q USERS WS1RIO3
```

```
Q USERS WS2RIO3
Q USERS WS3RIO3
Q USERS WS4RIO3
Q USERS LB5RIO3
Q USERS QLS1RIO3
Q USERS QLS2RIO3
```

Entire site

Startup:

```
XAUTOLOG LB1RIO3
XAUTOLOG LB2RIO3
XAUTOLOG LB3RIO3
XAUTOLOG LB4RIO3
XAUTOLOG WS1RIO3
XAUTOLOG WS2RIO3
XAUTOLOG WS3RIO3
XAUTOLOG WS4RIO3
XAUTOLOG WS5RIO3
XAUTOLOG WS6RIO3
XAUTOLOG WS7RIO3
XAUTOLOG WS8RIO3
XAUTOLOG LB5RIO3
XAUTOLOG LB6RIO3
XAUTOLOG QLS1RIO3
XAUTOLOG QLS2RIO3
XAUTOLOG QLS3RIO3
XAUTOLOG QLS4RIO3
```

Check if up:

```
Q USERS LB1RIO3
Q USERS LB2RIO3
Q USERS LB3RIO3
Q USERS LB4RIO3
Q USERS WS1RIO3
Q USERS WS2RIO3
Q USERS WS3RIO3
Q USERS WS4RIO3
Q USERS WS5RIO3
Q USERS WS6RIO3
Q USERS WS7RIO3
Q USERS WS8RIO3
Q USERS LB5RIO3
```

```
Q USERS LB6RIO3
Q USERS QLS1RIO3
Q USERS QLS2RIO3
Q USERS QLS3RIO3
Q USERS QLS4RIO3
```

Table 4.11: Scenario III Administrate: The non-MLN approach

	One	**Half**	**All**
Startup			
Number of commands issued	1	9	18
Number of characters used on commands	16	146	292
Check if up			
Number of commands issued	1	9	18
Number of characters used on commands	15	137	274

The MLN approach

Create

LINUX1 is logged on to and the entire creation process takes place form this system. When creating the site with MLN, the whole architecture is described in the project file. Note that because every VM in this scenario has more than one network interface card, the autoenum plugin cannot be used. In its current state autoenum only supports one NIC per VM.

First the project file is created
`joe scenario3.mln`
Because of the size of the file, it is appended to this report as Appendix I
Then the project is built
`mln build -f scenario3.mln`
The site is now ready to be used

Table 4.12: Scenario III Create: The MLN approach

Number of commands issued	2
Number of characters used on commands	43
Number of lines written to file	141
Number of characters written to file	2217
Number of lines copied	190
Number of files edited	1
Number of systems logged on to	1

Administrate

Since this scenario contains eighteen VMs, the "half of the VMs " will consist of the nine VMs on the left half of Figure 4.3. In the MLN approach, the administrative tasks takes place in Linux.

One VM

Startup:

```
mln start -p scenario3 -h ws1
```

Check if up:

```
mln status -p scenario3 -h ws1
```

Half of the VMs

Startup:

```
mln start -p scenario3 -h lb1
mln start -p scenario3 -h lb2
mln start -p scenario3 -h ws1
mln start -p scenario3 -h ws2
mln start -p scenario3 -h ws3
mln start -p scenario3 -h ws4
mln start -p scenario3 -h lb5
mln start -p scenario3 -h sqls1
mln start -p scenario3 -h sqls2
```

Check if up:

```
mln status -p scenario3 -h lb1
mln status -p scenario3 -h lb2
mln status -p scenario3 -h ws1
mln status -p scenario3 -h ws2
mln status -p scenario3 -h ws3
mln status -p scenario3 -h ws4
mln status -p scenario3 -h lb5
mln status -p scenario3 -h sqls1
mln status -p scenario3 -h sqls2
```

Entire site

Startup:

```
mln start -p scenario3
```

Check if up:

```
mln status -p scenario3
```

Table 4.13: Scenario III Administrate: The MLN approach

	One	**Half**	**All**
Startup			
Number of commands issued	1	9	1
Number of characters used on commands	29	265	22
Check if up			
Number of commands issued	1	9	1
Number of characters used on commands	30	274	23

4.7 Analysis of the collected metrics

The metrics collected in the scenario will now be analyzed each by themselves, and in the end a summary and overall impression will be presented.

4.7.1 Number of commands issued

Table 4.14: Number of commands issued

	non-MLN	MLN	Factor
Scenario I	22	2	11
Scenario II	116	2	58
Scenario III	500	2	250

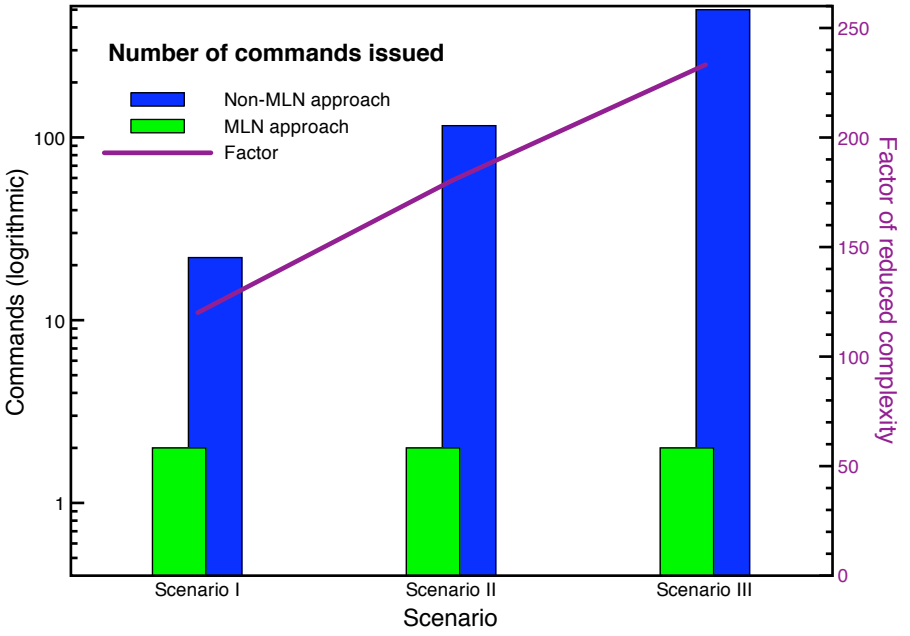

Figure 4.4: *Number of commands issued*

In the non-MLN approach, the metric shows interesting characteristics. If compared against the number of VMs in each scenario: As the scenarios contains 1, 6 and 18 VMs and the number of commands are 22, 116 and 500, we can see that it is not linear.

$$\frac{22}{1} = 22$$

$$\frac{116}{6} = 19.33$$

$$\frac{500}{18} = 27.78$$

We see that the number of commands per VM decreases after the first scenario. That is to be expected since the extra VMs after the first does not need to go through the z/VM commands once more. From scenario II to scenario III there is an increase in in the average. The reason for this is the increased network complexity, and this is an important point considering metrics of this nature. It is often easy to forget that the increased complexity is mainly because of the networking aspect of the virtual system. It appears that the number of commands issued will increase in a way similar to

$$O(v \cdot s)$$

where v is the number of VMs and s is the number of switches.
When both v and s becomes large

$$O(v \cdot s) \rightarrow O(n^2)$$

If there are no network topology, the VMs simply being connected to the outgoing switch as in "Scalable deployment and configuration of high-performance virtual clusters" [7], the increase of the number of VMs is predicted to increase the metric in a more linear pattern.

$$O(v)$$

As will be shown later, the average of characters per command also increases with the size and complexity of the scenario. This indicates that the command aspect of the non-MLN approach scales poorly when the size and complexity of a site increases.

In the MLN approach, it is easy to se the trend and development as the number of commands stays constant. Unless the complexity of the two commands increases notably then this indicates that commands in this approach are unaffected by size and complexity of the scenario. As can be seen from "Number of characters used on commands", the number of characters stays constant as well. It can therefore be argued that commands in this approach scales perfectly.

When comparing the two approaches based on commands, it is quite clear that MLN comes out as the superior solution. Since MLN stays constant while non-MLN increases rapidly, the number of commands with non-MLN increases by an increasingly rising factor compared to the MLN approach.

4.7.2 Number of characters used on commands

It is interesting to observe the data from the non-MLN approach. It appears that the number of characters used on commands increases in a non-linear manner when the size and complexity increases. As will be seen later, the average number of characters per command increases as well. An explanation to this may be that longer, more complex commands outweighs smaller ones to a greater extent when the complexity and size of a site increases. With this metric we see further proof that this aspect of the non-MLN approach scales badly with increased size and complexity.

As with "Number of commands issued", "Number of characters used on commands" stay constant in the MLN approach. This strengthens the credibility of perfect scalability when considering commands.

Table 4.15: Number of characters used on commands

	non-MLN	MLN	Factor
Scenario I	448	43	10.42
Scenario II	3066	43	71.3
Scenario III	16675	43	387.79

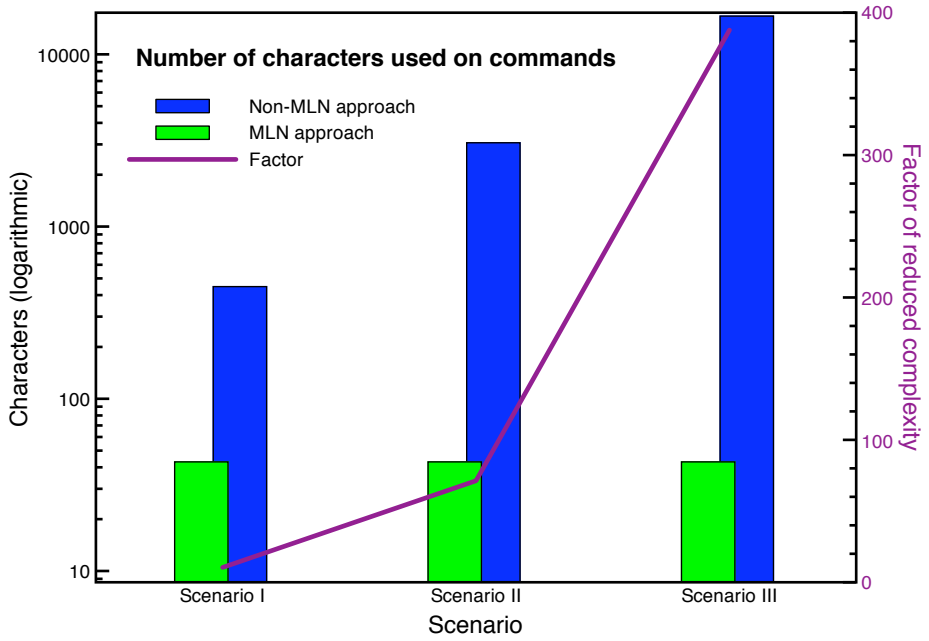

Figure 4.5: *Number of characters used on commands*

Once again the factor of reduction of complexity increases dramatically since the MLN approach remains constant while the non-MLN approach increases at a considerable rate.

4.7.3 Number of lines written to file

Table 4.16: Number of lines written to file

	non-MLN	MLN	Factor
Scenario I	24	15	1.6
Scenario II	67	48	1.4
Scenario III	245	141	1.74

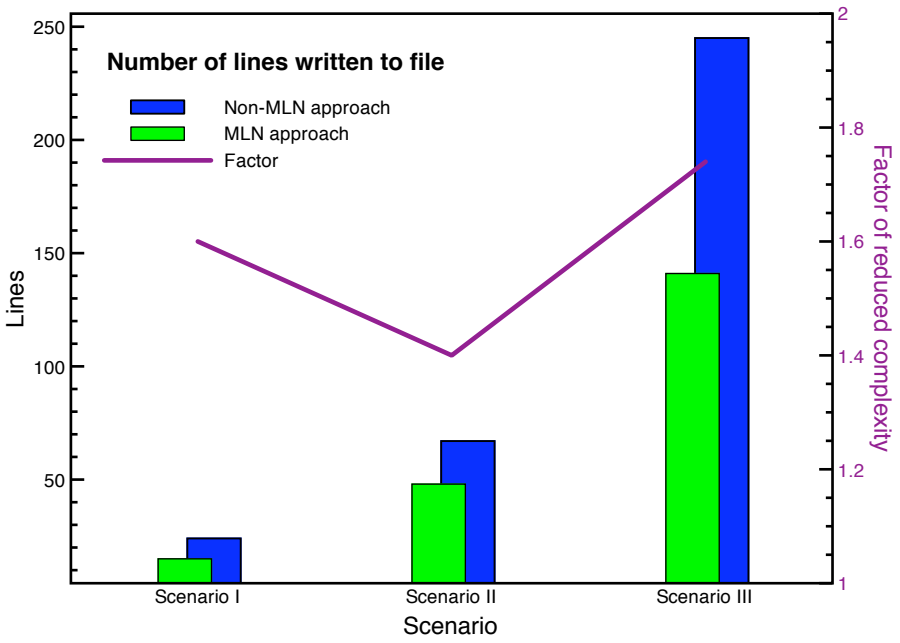

Figure 4.6: *Number of lines written to file*

In the non-MLN approach we see the same pattern as with "Number of commands issued". This was expected since the first VM has to set several attributes and settings that will be used by later VMs. Also, for the first VM, all lines has to be written. No lines can be copied since there are nowhere to copy them from.

It is once again probable that the factor of which lines increase from scenario II to scenario III is because of the increased complexity of the network. This is mainly because of the definition and configuration of the additional

Table 4.17: Non-MLN Lines increase factor

	Factor
Scenario I -> Scenario II	2.79
Scenario II -> Scenario III	3.66
Scenario I -> Scenario III	10.21

network interface cards in both z/VM and Linux.

In the MLN approach we also observe a distinct rise in the number of lines written, although not as steep as with the non-MLN approach. We calculate the increase factor to derive more information.

Table 4.18: MLN Lines increase factor

	Factor
Scenario I -> Scenario II	3.2
Scenario II -> Scenario III	2.94
Scenario I -> Scenario III	9.4

The cause of the factor decrease from 3.2 to 2.94 is the high reusability of code, as can be seen by "Number of lines copied", and the use of super-classes. It is features like these that help the scalability of this approach. Also we see the affect of the network complexity has a reduced effect as the MLN approach automatically handles the configuration of the Linux level network interfaces.

When comparing the two approaches, we see that the MLN approach comes out with less lines in all scenarios. The factor of difference is not constant but shows that the MLN offers an increased reduction on the first VM created compared to the immediately following. It then increases the gain as the size and complexity increases.

4.7.4 Number of characters written to file

Table 4.19: Number of characters written to file

	non-MLN	**MLN**	**Factor**
Scenario I	550	208	2.64
Scenario II	1680	736	2.28
Scenario III	6788	2217	3.06

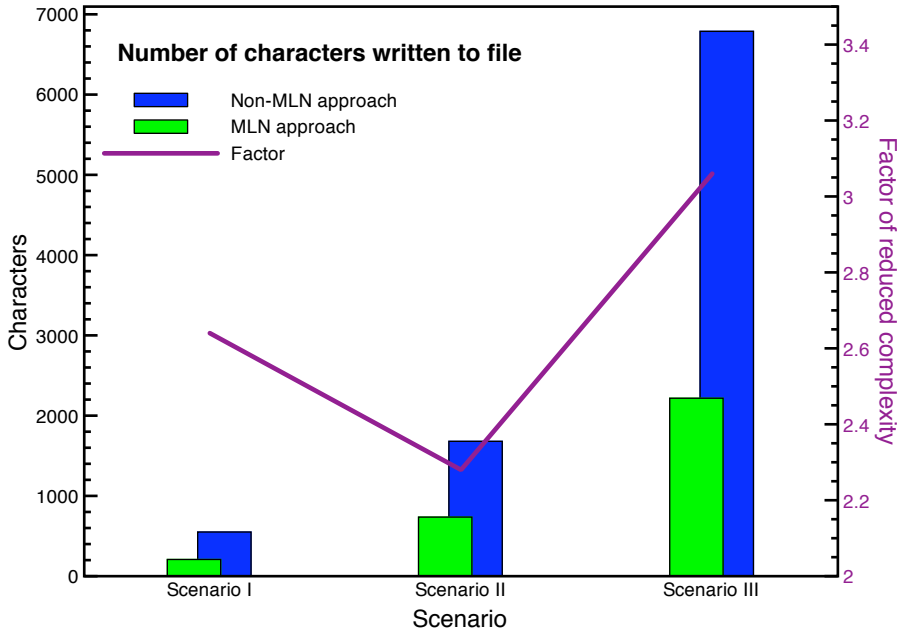

Figure 4.7: *Number of characters written to file*

In the non-MLN approach the characters written to file increases at a higher rate than lines written to file as can be seen in Table 4.20.

Table 4.20: Non-MLN Lines and Character increase factor

	Factor	
	Lines	**Characters**
Scenario I -> Scenario II	2.79	3.05
Scenario II -> Scenario III	3.66	4.04
Scenario I -> Scenario III	10.21	12.34

This tells us that the length, and therefore probably the complexity, of the lines increases as the size and complexity of the scenario increases. This is further supported by the average number of characters per written line in section 4.7.9.

In the MLN approach we see the same trend as in the non-MLN approach as shown in Table 4.21.

As is the case in the non-MLN approach, this tells us that the length, and therefore probably the complexity, of the lines increases as the size and complexity of the scenario increases. This is also further supported by the average number of characters per written line in section 4.7.9.

Table 4.21: MLN Lines and Character increase factor

	Factor	
	Lines	**Characters**
Scenario I -> Scenario II	3.2	3.54
Scenario II -> Scenario III	2.94	3.71
Scenario I -> Scenario III	9.4	10.66

This is in accordance with the experience from the scenarios. As a frequently copied line was "}", these had a smaller impact on the metric when the size and complexity increased.

When comparing the two approaches, we see that the factor of increased characters starts off at 2.64, then drops to 2.28 before it increases beyond the previous two at 3.06. It is also apparent from Table 4.20 and Table 4.21 that the non-MLN approach has a higher factor of increased line and character number except when going from Scenario I to Scenario II.

4.7.5 Number of lines copied

Table 4.22: Number of lines copied

	non-MLN	**MLN**	**Factor**
Scenario I	0	0	0
Scenario II	7	34	4.86
Scenario III	60	190	3.17

In the non-MLN approach we see that the number of lines starts on zero, as there is nowhere to copy the lines from. It then starts out low at 7 in Scenario II before it increases dramatically in Scenario III. From Scenario II to Scenario III the number increases with a factor of 8.57, hinting that the reuse of code in the non-MLN approach scales impressively with increased size and complexity of the scenario. It does however not tell us all that much before we take the total number of lines written to file into consideration.

When calculating the percentage of the total number of lines that were copied using "Number of lines written to file" and "Number of lines copied", we get

$$\frac{100\%}{67+7} \cdot 7 = 9.46\%$$

for Scenario II, and

$$\frac{100\%}{245+60} \cdot 60 = 19.67\%$$

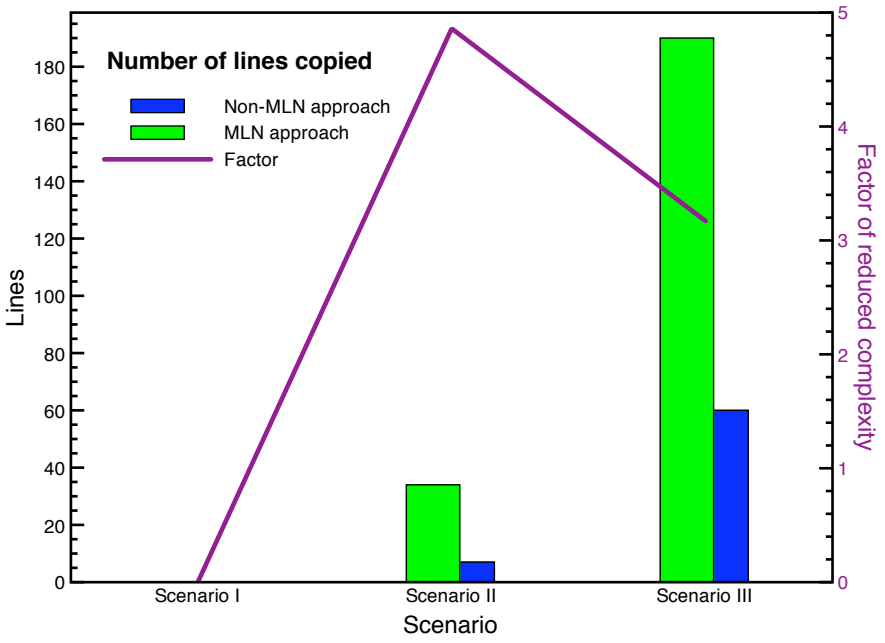

Figure 4.8: *Number of lines copied*

for Scenario III. We can indeed see that it has been possible to reuse a larger part of the lines.

In the MLN approach, as with the non-MLN approach, the number starts at zero. It the increases to 34 in Scenario II and ends up at 190 in Scenario III. From scenario II to Scenario III it increases by a factor of 5.59. To get a more accurate picture, we calculate what percentage of the total written lines where copied. Starting with Scenario II, we get

$$\frac{100\%}{48+34} \cdot 34 = 41.46\%$$

and for Scenario III, we get

$$\frac{100\%}{141+190} \cdot 190 = 57.4\%$$

We see here that the MLN approach is able to reuse an impressive number of the lines. In Scenario III, more than half of the lines are actually reused.

Comparing the two approaches, we see that the non-MLN approach increases its number of copied lines with a much higher factor than the MLN approach. However, when we take into account the total number of lines, and what percentage of this has been reused; the MLN approach shows a higher aptitude for reusing code.

4.7.6 Number of files edited

Table 4.23: Number of files edited

	non-MLN	MLN	Factor
Scenario I	4	1	4
Scenario II	23	1	23
Scenario III	126	1	126

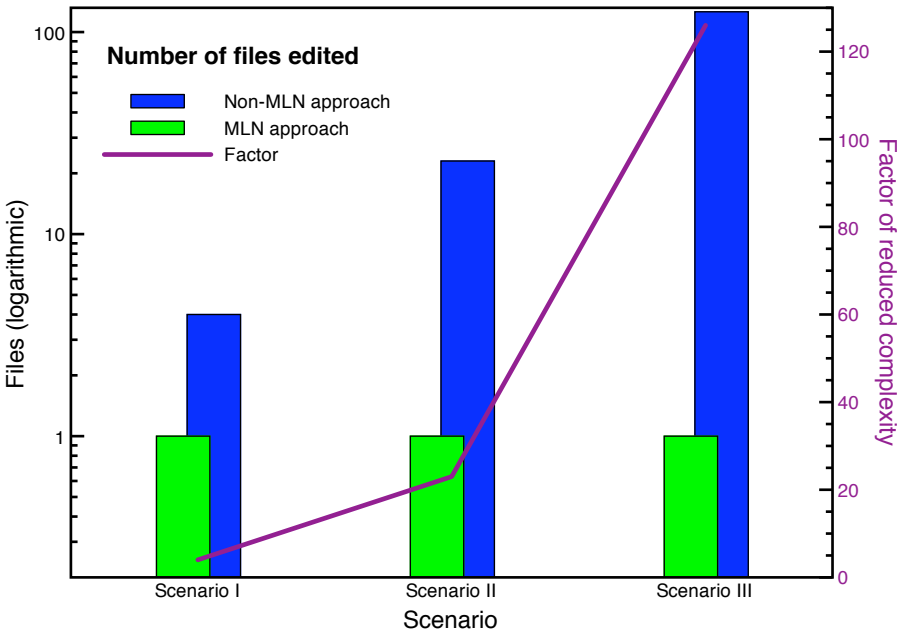

Figure 4.9: *Number of files edited*

In the non-MLN approach, distinct pattern can be deduced by considering what the files are used for. In any project, two files are always edited in z/VM (one of these files, PROFILE EXEC, only has to be created once on the system and will be reused by future projects). On the Linux side, a minimum of two files are edited. One for setting the hostname, and one for configuring the network interface card with IP address and so on. Subsequently, any additional files edited are for additional NICs. This metric is therefore largely dependent on the complexity of the network. When the number of NICs becomes much greater than the number of VMs, this metric approaches

$$O(n)$$

where n is the number of NICs in the scenario. It also approaches this form

if the number of VMs increases while networking is limited to their direct connection to the outgoing switch. In this case n is the number of VMs in the scenario.

In the MLN approach, there is always only one file to consider, the MLN project file. As with the other metrics that stays constant with this approach, it indicates that it scales extremely well on this point when the size and complexity increases. It also makes the operations related to the manipulation of files extremely centralized. It can be argued that this has the drawback of making the one file big and difficult to keep track of when the size and complexity of the scenario increases.

Since the number of files in the MLN approach is constant while the number of files in the non-MLN approach continues to rise when the size and complexity of the scenario increases, the factor of reduction increases with the same speed as the number of files edited in the non-MLN approach.

4.7.7 Number of systems logged on to

Table 4.24: Number of systems logged on to

	non-MLN	**MLN**	**Factor**
Scenario I	2	1	2
Scenario II	2	1	2
Scenario III	2	1	2

"Number of systems logged on to" is probably the least informative metric measured in the scenarios. Both approaches therefore seems to scale perfectly on this point as the size and complexity of the scenarios have absolutely no effect on the metric.

4.7.8 Average number of characters per command

Average number of characters per command is calculated by

$$\frac{Number\ of\ characters\ used\ on\ commands}{Number\ of\ commands\ issued} \tag{4.1}$$

The average number of characters per command were calculated to give a better impression of the average complexity of the commands.

In the non-MLN approach, we see that the average number of characters per command increases by each scenario. This is an indication that the

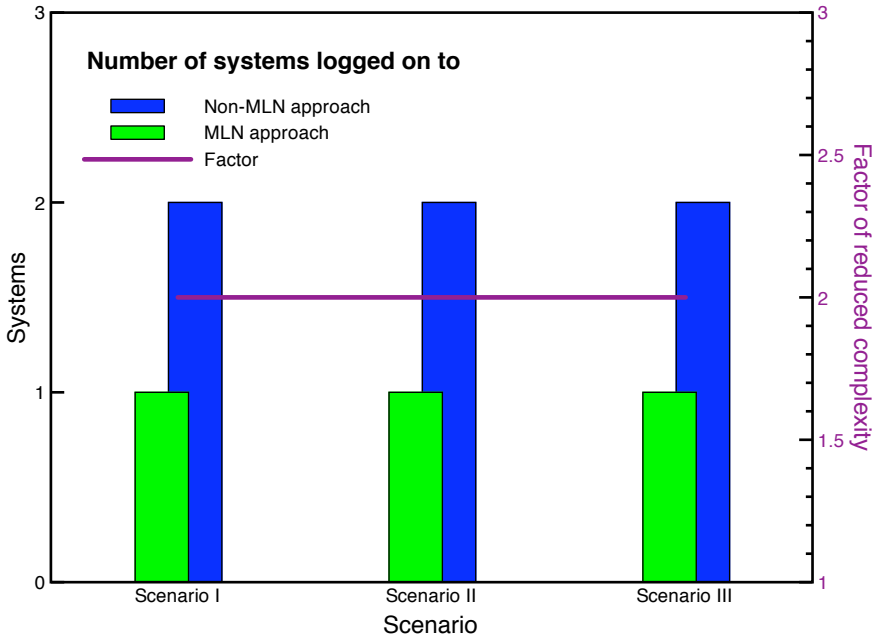

Figure 4.10: *Number of systems logged on to*

Table 4.25: Average number of characters per command

	non-MLN	**MLN**	**Factor**
Scenario I	20.36	21.5	0.95
Scenario II	26.43	21.5	1.23
Scenario III	33.35	21.5	1.55

length (and thereby complexity) of the commands increases with the increased size and complexity of the scenarios. It can also indicate that as the size and complexity of the scenarios increases, the number of long commands increases at a greater speed than the number of short commands. To derive more information, we calculate the increase factor between the scenarios.

Table 4.26: Non-MLN Average number of characters per command increase factor

	Factor
Scenario I -> Scenario II	1.3
Scenario II -> Scenario III	1.26
Scenario I -> Scenario III	1.64

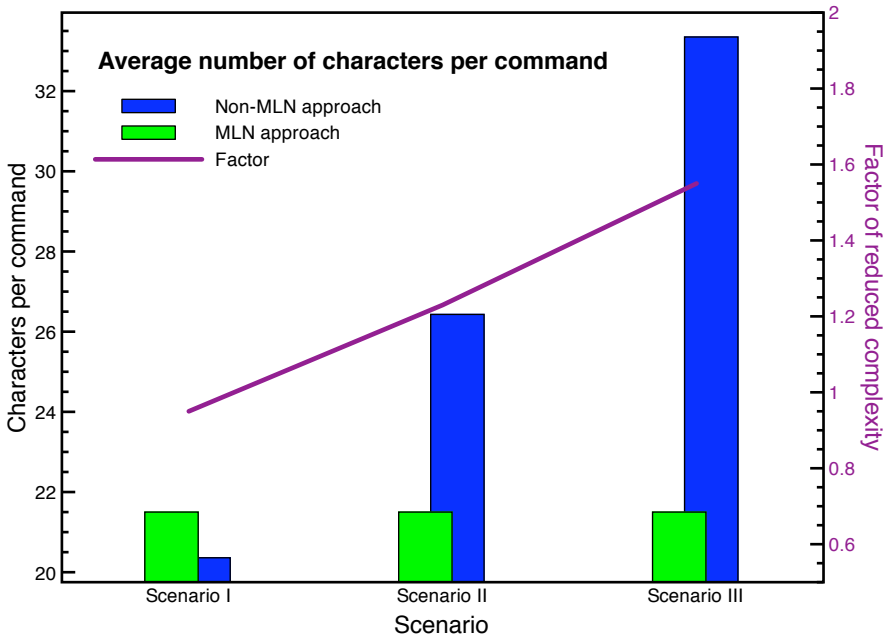

Figure 4.11: *Average number of characters per command*

From these calculations we see that, although the average number of characters per command increases, the rate of which it increases decreases between the scenarios. This can indicate that the factor flattens out as the size and complexity of the scenarios increases.

In the MLN approach the average number of characters per command stays constant. This is hardly surprising as the metrics used to calculate the average are both constants for the MLN approach.

When comparing the non-MLN approach with the MLN approach, we see the same trend as previously when the metric of one approach stays constant. The non-MLN approach starts out with a lower average than the MLN approach in Scenario I; then it increases beyond the MLN approach while the MLN approach stays constant. The factor of reduced complexity rises as the average number of characters per command for the non-MLN approach increases.

4.7.9 Average number of characters per written line

Average number of characters per written line is calculated by

$$\frac{Number\ of\ characters\ written\ to\ file}{Number\ of\ lines\ written\ to\ file} \tag{4.2}$$

Table 4.27: Average number of characters per written line

	non-MLN	MLN	Factor
Scenario I	22.92	13.87	1.65
Scenario II	25.07	15.33	1.64
Scenario III	27.71	15.72	1.76

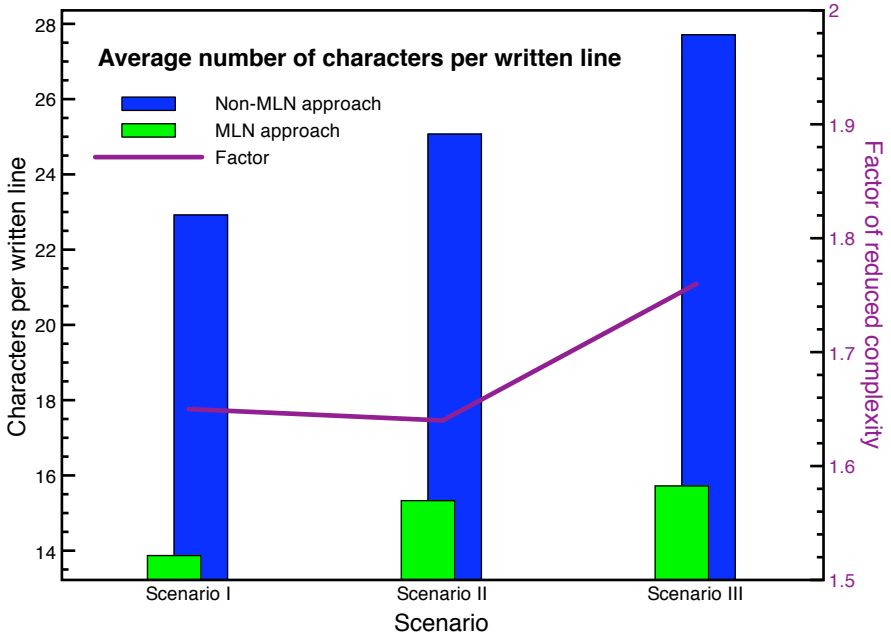

Figure 4.12: *Average number of characters per written line*

Like "Average number of characters per command", "Average number of characters per written line" was calculated to give an indication of the complexity of the lines.

In the non-MLN approach, we see a steady rise in the calculated average. It is comparable to the "Average number of characters per command" in that it indicates that the length (and thereby complexity) of the lines increases with the increased size and complexity of the scenarios. It can also that as the size and complexity of the scenarios increases, the number of long lines increases at a greater speed than the number of short lines. This can be because a greater number of the shorter lines could be copied instead of being written. To derive more information, we calculate the increase factor between the scenarios.

The factor with which the average increases between the scenarios appears to be steadily rising. This is consistent with the behavior of the met-

Table 4.28: Non-MLN Average number of characters per line increase factor

	Factor
Scenario I -> Scenario II	1.09
Scenario II -> Scenario III	1.11
Scenario I -> Scenario III	1.21

rics from which the average was calculated.

In the MLN approach we also see a trend similar to the non-MLN approach much for the same reason. The number of short lines does not increase with the same speed as the number of long lines when the size and complexity of the scenarios increases. A significant reason for this is that the frequency of which short lines like "}" are copied appears to increase as the size and complexity of the scenarios increases. We calculate the increase factor between the scenarios.

Table 4.29: MLN Average number of characters per line increase factor

	Factor
Scenario I -> Scenario II	1.11
Scenario II -> Scenario III	1.03
Scenario I -> Scenario III	1.13

Interestingly, the factor decreases when going from Scenario II to Scenario III compared to going from Scenario I to Scenario II. This might indicate that the factor with which the average number of characters per line increases with as the size and complexity of the scenario increases decreases. If this is indeed the case, then it would a promising trend when considering scalability.

When comparing the two approaches, the MLN approach appears to behave most promisingly. Not only is the average number of characters per line lower, but the factor of reduced complexity increases from Scenario II to Scenario III after having changed minimally between Scenario I and Scenario II.

4.7.10 Average number of lines per file

Average number of lines per file is calculated by

$$\frac{Number\ of\ lines\ written\ to\ file\ +\ Number\ of\ lines\ copied}{Number\ of\ files\ edited} \tag{4.3}$$

Table 4.30: Average number of lines per file

	non-MLN	MLN	Factor
Scenario I	6	15	2.5
Scenario II	3.22	82	25.47
Scenario III	2.42	331	136.78

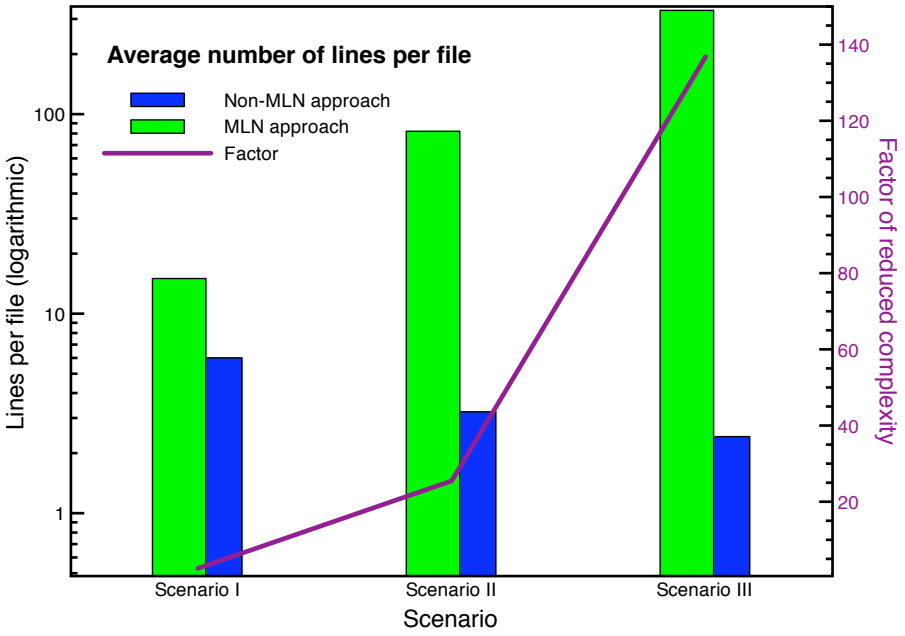

Figure 4.13: *Average number of lines per file*

In the non-MLN approach we see a decrease in the average number of lines per file. The reason for this is that each of the files changed on the Linux side usually only has one line manipulated. This means that if the complexity of the scenario increases at a greater rate than the size, the average number of lines per file will decrease. This is because the creation and configuration of the VSWITCHes happens outside of files. The network interface cards of the hosts are the only network parts defined in files.

As the MLN approach only uses one file, the average number of lines per file is simply the number of lines in this one file. For more information, we calculate the increase factor between the scenarios.

The factor of which the file increases between scenarios is relatively big. Although it decreases from 5.47 to 4.04 when looking at the single jumps between scenarios, we see that it increases with as much as 22.07 when go-

Table 4.31: MLN Average number of lines per file increase factor

	Factor
Scenario I -> Scenario II	5.47
Scenario II -> Scenario III	4.04
Scenario I -> Scenario III	22.07

ing from Scenario I to Scenario III. Something to take note of is that the project file for Scenario I contained 3 lines with just "}" while the project file for Scenario III contained 94 lines of the same type. This type of line therefore has a surprising impact on this calculated metric.

When comparing the two approaches, the MLN approach clearly has the highest metric. And while value for the MLN approach rises, the value for the non-MLN approach decreases. It is interesting to consider which one is favorable. The non-MLNs approach low and decreasing value indicates low complexity in the changes made to the files, while the MLN approach centralizes the information in one file. The preferred method here depends somewhat on personal preferences. When considering that the high number of files accessed in the non-MLN approach resides on all the different Linux VMs, the task of getting to the files becomes much more cumbersome than in the MLN approach. It therefor seems logical to favor the single-file solution.

4.7.11 The administrational metrics

We will now take a look at the metrics form the administrational tasks "startup" and "check if up". Because of the uniform trend in the metrics, they will be analyzed in groups.

4.7.12 Startup

Table 4.32: Scenario I Startup: Commands issued

	Non-MLN	MLN
One	1	1
Half	1	1
All	1	1

Table 4.33: Scenario I Startup: Characters used on commands

	Non-MLN	MLN
One	16	29
Half	16	29
ALL	16	22

Table 4.34: Scenario II Startup: Commands issued

	Non-MLN	MLN
One	1	1
Half	3	3
All	6	1

Table 4.35: Scenario II Startup: Characters used on commands

	Non-MLN	MLN
One	16	29
Half	48	87
All	96	22

Table 4.36: Scenario III Startup: Commands issued

	Non-MLN	MLN
One	1	1
Half	9	9
All	18	1

Table 4.37: Scenario III Startup: Characters used on commands

	Non-MLN	MLN
One	16	29
Half	146	265
All	292	22

We can see from these metrics that the two approaches handles the task of starting VMs in the same manner, except for in one exception. In both approaches startup commands are issued to the individual VMs. The only difference between non-MLN and MLN being the length of the command used. Because of this, the task increases in workload according to

$$O(n)$$

Where n is the number of VMs being started. The only other variable it the length of the VM and MLN project names (The effect of this can be seen in the metrics). Since the non-MLN approach uses a shorter command and only uses a maximum eight character name, it can be seen as more scalable than the MLN approach when dealing with individual VMs. The notable exception is when all of the VMs in the scenario are to be started. In this case the non-MLN approach starts every VM individually, while the MLN approach runs one single command to start the whole project. This means that for the MLN approach, the size of the scenario has no effect on the size of the task of starting all the VMs.

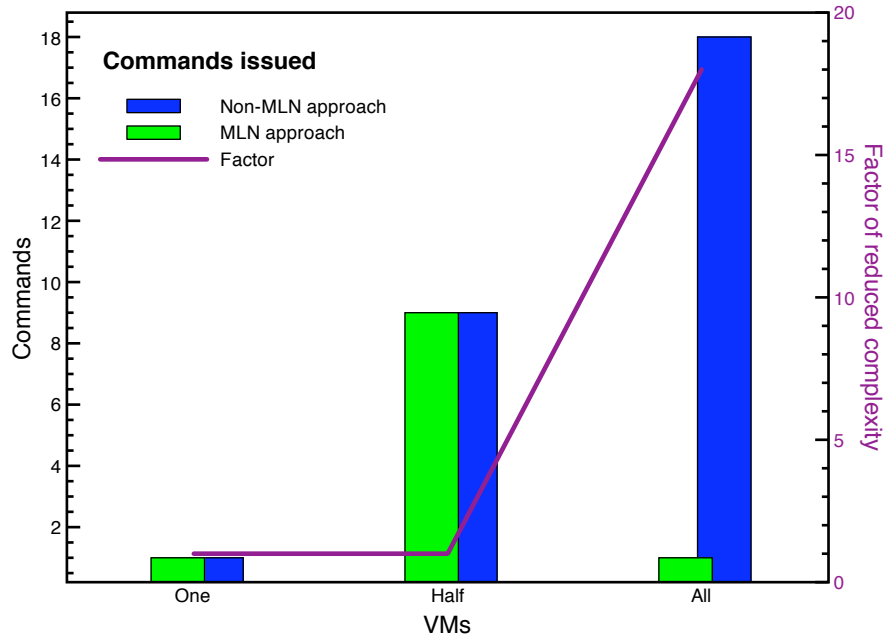

Figure 4.14: *Scenario III Commands issued on startup task*

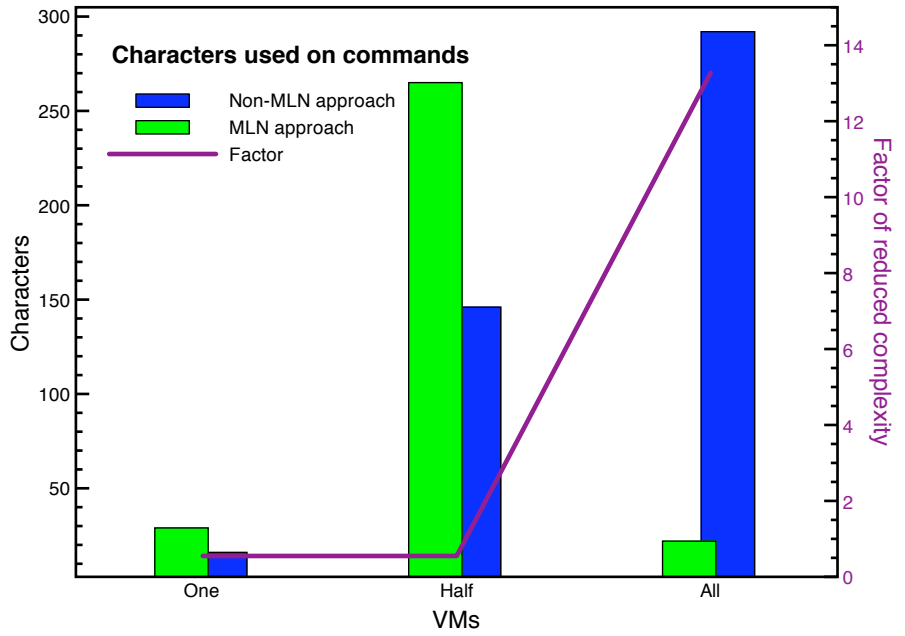

Figure 4.15: *Scenario III Characters used on commands during startup task*

4.7.13 Check if up

Table 4.38: Scenario I Check if up: Commands issued

	Non-MLN	MLN
One	1	1
Half	1	1
All	1	1

Table 4.39: Scenario I Check if up: Characters used on commands

	Non-MLN	MLN
One	15	30
Half	15	30
All	15	23

Table 4.40: Scenario II Check if up: Commands issued

	Non-MLN	MLN
One	1	1
Half	3	3
All	6	1

Table 4.41: Scenario II Check if up: Characters used on commands

	Non-MLN	MLN
One	15	30
Half	45	90
All	90	23

Table 4.42: Scenario III Check if up: Commands issued

	Non-MLN	MLN
One	1	1
Half	9	9
All	18	1

Table 4.43: Scenario III Check if up: Characters used on commands

	Non-MLN	MLN
One	15	30
Half	137	274
All	274	23

When looking at the results from the task "check if up" we see that it follows the exact same pattern as the task "startup". As long as specific VMs are being checked, the size of the task increases according to

$$O(n)$$

Where n is the number of VMs being checked.

The notable exception being when checking all the VMs in the scenario in the MLN approach. In that case MLN can check all the VM with a single command. This makes the workload of that particular case constant and independent from the number of VMs being checked.

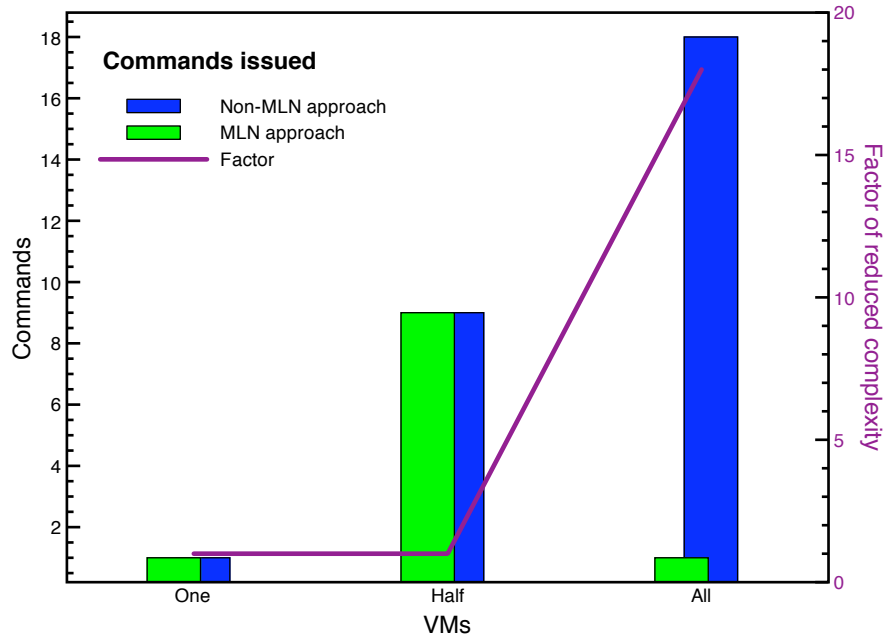

Figure 4.16: *Scenario III Commands issued on check if up task*

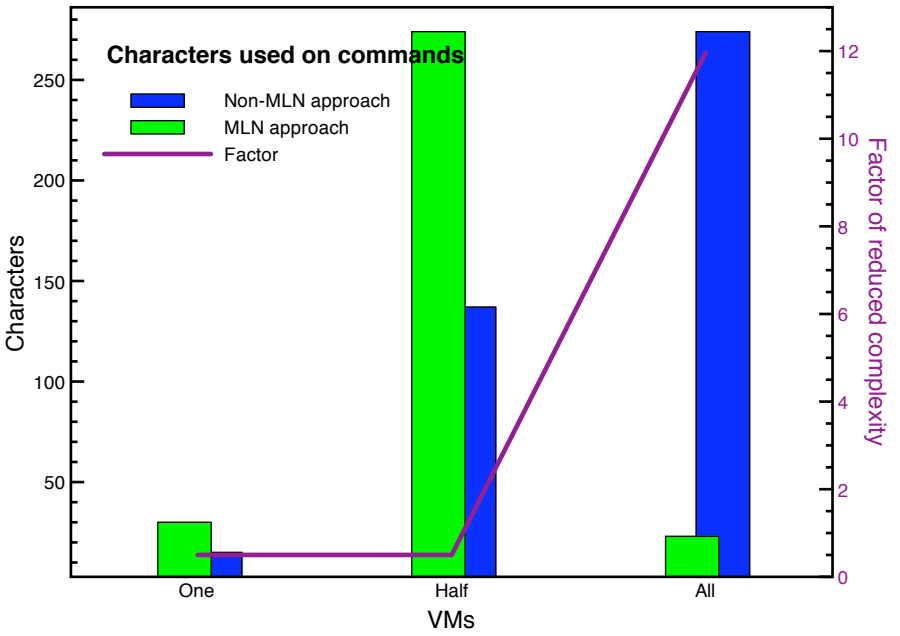

Figure 4.17: *Scenario III Characters used on commands during check if up task*

4.7.14 Overall metrics summary

Looking back at the metrics as a whole, it becomes apparent that the MLN approach shows improvements in most of the cases. Most notably is perhaps the number of files edited (Table 4.23) and the number of commands issued (Table 4.14). The significant advantage of the reduction of files edited becomes more apparent when considering the wide dispersion of files over the VMs in the non-MLN approach. The number of commands issued also indicates that the MLN approach scales better with increased network complexity, much thanks to the automation of VSWITCH creation and configuration on the z/VM layer. The areas where the non-MLN shows better scalability are mostly in the administration section. More specifically, the complexity of the commands issued when working with individual guests (Table 4.15 and Table 4.17), the exception in these cases being when the entire site is administrated.

In addition, some of the metrics indicates a predictable trend. Most notably in number of commands issued (Table 4.14) and number of files edited (Table 4.23). For some of the others however, it is more difficult to see the trend, examples being the number of lines copied (Table 4.22) and the number of characters written to file (Table 4.19).

Chapter 5

Discussion

The purpose of this chapter is to discuss the finished system and the process that the project has gone through. The final architecture will be looked at and compared to the originally planned architecture in section 5.1. The possibility for replicating the system and the results from this project is discussed in section 5.2 and alternative approaches will be considered in section 5.3. The experience with MLN is outlined in section 5.4, and possibilities for further development of MLN is also explored. Alternative problem statements not chosen are mentioned in section 5.5. The validity of the scenarios and the metrics measured are discussed in section 5.6. Some limitations are observed and further testing proposed. Compromises made during the project are addressed in section 5.7, as well as their cause. Finally, experience and knowledge acquired during the project, along with some advice, brings the chapter to an end with section 5.8.

5.1 The Final Architecture Complexity

The final architecture did not contain any additional major modules other than those planned from the beginning. So on the high level of the architecture, the complexity did not supersede that which was expected. On the lower level however, the situation is somewhat different. Some of the modules, more specifically DirMaint, were not exactly plug-and-play. Although DirMaint is not to complicated to set up and use if it works as intended all the time. However, considerable more knowledge is needed if an error occurs. This is of course true for all the modules, but it applies especially to DirMaint, as its workings are supposed to be completely closed and automated when set up.

The level of necessary z/VM knowledge assumed in the beginning of the project can therefore be perceived as somewhat naive. Not only is a certain familiarity with z/VM necessary, but experience and knowledge about DirMaint beyond simply installing it can with a high degree of certainty be

seen as a necessity. This is of course only a requirement for the administrator(s) directly responsible for the z/VM layer.

5.2 Replication

In a practical project like this, meant to be a guide, starting point or even just an encouragement for others that might find the topic interesting, replication is critical. The ability to recreate the results of this endeavor, to use it or develop it further. In this case, it is relatively safe to claim that replication should not be a problem. If the necessary resources are available, more specifically access to certain IBM hardware and software, no other requirements needs to be satisfied. All code and configurations created during this project are freely available and should be supplied with this text. The nature of the metrics measured in the scenarios makes them predictable and constant in the closed environment of the system, therefore a replicated system should produce the same results.

5.3 Alternative Approaches

Looking back at the project with hindsight and reconsidering the alternatives and choices made regarding the architecture in the design phase, no significant changes would be made should the choices arise again. However, had time not been the limiting factor, more resources would be delegated to finding an alternative to DirMaint. One could also consider seriously to look into the possibility of developing a disk space management module specifically for the system. The main reason this would be desirable is to remove the only non-free module in the system, but also to move as much as possible of the system to open source.

5.4 Adding virtualization platforms to MLN

All in all, MLN was quite friendly when it came to modifications as it did not require extensive modification to its pre-existing architecture and code. The hooks that were added were not specified toward the z/VM plugin, but has the possibility to be used by plugins in general. Because of this, the project has had a beneficial but somewhat unexpected side effect in that MLN is now much more capable of using future plugins without modifications to its core code. It should therefor be easier to add support for additional virtualization platforms in the future.

That being said, the changes might give the impression of being "slapped on" as an afterthought (hardly surprising since that was what actually happened). This is mostly because of the hardcoded, virtualization platform

specific code embedded in MLN. Although the solution and approach in this project worked nicely this time, it is hard to say if this method will scale. If every new plugin just have to make "a few modifications" to MLNs base code, MLN might turn out to consist of clumsy and bloated code. This approach seems somewhat backwards and counter productive. Therefore, if MLN is to increase its base of supported virtualization platforms, more drastical changes to MLNs base code and architecture can prove beneficial.

One solution would be to make MLN itself completely platform independent. The platform specific code (the code for UML and Xen) would then be moved to plugins, preferably one plugin per platform. MLN could then have a strict, standardized way of using these plugins thereby making the plugins adapt to MLN and not the other way around. This way, MLNs core code could be kept clean and efficient.

The z/VM plugin created in this project can be seen as a first step in this direction. The base code of MLN does not contain any code specifically written for z/VM, and z/VM therefor becomes the very first virtualization technology that MLN can use while being completely unaware of its existence. This is to a high degree why it should be relatively easy to reuse this approach for other virtualization technologies, and can therefore be seen as an important proof-of-concept for the possibility of making MLN technology-independent. It is not unreasonable to predict that the experience from this project can be of great help if MLN is to be further developed in this direction.

5.5 Alternative Problem Statements

Alternative problem statements to this project could have involved making and using only open source, and not using any non-free software. As this would have been an added constraint, it is likely that the time needed to complete the project would have increased. Another possible problem statement would be to create a more user friendly and automated front end in z/VM itself, this would however shift the project in a completely different direction as the Linux part would be completely removed. This would also have the effect that the new guests could not have their filesystem altered from the system.

5.6 Validity of the scenarios and metrics

Looking back at the scenarios and the results obtained from them, one can ask if they were objectively chosen. Certainly the majority of the results speak in favor of the new system. It is important to keep in mind however, the main strength of MLN: To handle a great number of identical VMs. In

none of the scenarios was this trait used to its fullest as features like autoenum could not be used. Had more time been available, more scenarios tailored for the two different approaches could have been conducted. In this report however, the scenarios were an attempt to be a neutral middle ground between the approaches so as not to favor any of them.

After a practical project like this, with goals like automation, simplification and easing a process, it is difficult to measure the result in simple metrical terms. How does one measure strain, or a persons impression of a task? The metrics chosen to be measured in the scenarios where chosen because they reasonably well could represent the workload for a system administrator. Some of them proved to be more reliable representations than other. Especially when considering the two metrics "number of lines written to file" and "number of lines copied" it is important to remember the definitions described earlier. Several lines in "number of lines written to file" were copied lines that just had a single numeric digit incremented, like names and IP addresses. Still, the metric reasonably represents the task of remembering to make the changes and not make an error when doing so.

Some might find it strange that no time factor was measured or calculated. The reason time was completely excluded as a metric is all the factors that affects time. Not only is the speed of the person typing a factor, but also at what speed the system completes commands. Estimates could be made, but these would most likely not be representative. Other factors that comes into play when evaluating scenarios like these are the users knowledge and familiarity with the system. It is unreasonable to assume that a normal user can type commands continually without needing to stop to think, consider or look up a command. It can be assumed, however, that when the diversity of commands increases, so does the likelihood for the need to look up information in manuals.

In the end, the scenarios were the best reasonable way of illustrating the change the new system has made to the administration environment. It would most likely have given more realistic and representative data if the two approaches were tested over a longer time period by a group of actual system administrators. In this project, that is however somewhat unrealistic. Not only are the human resources, hardware resources and time unavailable, but the system is not yet in a state where it should be used in a production environment. It would on the other hand be an interesting future follow-up project.

5.7 Compromises

During the project, compromises has been made in some areas. This has mainly been because of time constraints. The most notable areas are the robustness of the MLN plugin and the level of feedback from PROP and DirMaint to the administrating Linux guest.

The MLN plugin works as intended in its current state, but it is highly dependent on only receiving correct input and data. It contains very little checks to ensure that the commands it executes will not fail. This is because of the objective prioritization followed during the project: First make it work, then make it fault tolerant. The first part, make it work, has been achieved; the second part, make it fault tolerant, remains an ongoing process. The MLN plugin can therefore be described as currently existing in a beta state ready for publishing to the community for further testing.

The level of feedback from PROP and DirMaint to the administrative Linux guest is at this time non-existent. In praxis this means that the administrator must simply assume that the commands sent to PROP and DirMaint are executed successfully, no error messages will be received should a problem occur. The reason this has not been incorporated into the system is that, to the knowledge of the writer, no "official" solution to this situation exist. After talking to IBM experts in the mainframe area about the matter and consulting IBM documentation, it became apparent that to solve this problem would be quite time consuming. Although certainly something that should be added before the system is considered complete, it was not deemed realistic to incorporate it during this projects time frame.

5.8 Practical Experience

A lot of knowledge and experience has been gained through this project that is not covered or mentioned in this report. The reason it is not to be found in the report is because it is not necessarily relevant and because it would have made the report quite chaotic. Simple things like the fact that you can't just reboot a mainframe or even parts of it easily. Some situations are impossible to solve by yourself. For instance, in the early days of the project, I managed to shut down my entire z/VM environment with a single, unlucky command. There was no way I could do anything to solve this myself, all I could do was send an e-mail to the administrator (Malcolm Beattie) of the first level z/VM environment and ask for help. Luckily for me, he had my environment up and running in no time, and I was one important experience richer.

For anyone who considers embarking on a similar project, some advice can be given based on the experience gained during this period. First of all: Mainframes are a very specialized area, and the people who can give

quality help and advise are limited in numbers. In the end, we were lucky enough to have several people to turn to and all of them were incredible assets.

Secondly, don't be afraid to try things out. As with all new things, that's the way you learn. I probably learned the most about the workings of the system when I made errors and mistakes and had to fix them. Just be sure to know who to contact if one of those "impossible to solve by yourself" problems should appear.

Thirdly, time is a luxury and there will probably not be enough of it. Be sure to structure the work accordingly. It is therefore also important to plan ahead for emergencies.

Chapter 6

Conclusion

The primary goal of this thesis was to explore the possibility of simplifying the administration of guests in z/VM. The knowledge and expertise threshold for using the z/VM environment were considered to be to high for non-mainframe users. The scope was specified down to automating and abstracting the administration of Linux guests. It was planned from the beginning to use MLN as the administration tool and a plugin containing the z/VM specific code was developed. Changes were also made to the MLN base code to enable it to use plugins for technology specific code in general.

For the system to work, an underlying z/VM architecture had to be designed and created capable of working with MLN. This architecture had to be able to work independently without direct intervention from an administrator once the system was set up. The finished architecture consisted of three main modules:

- The administrating Linux guest running MLN with the plugin, used for administration of the environment
- The Programmable Operator, used mainly for security and safety
- The Directory Maintenance Facility, used mainly for storage management

When the system was operational, it was used in three scenarios and compared with a default z/VM environment. Several metrics were measured in each scenario to indicate the complexity of the tasks and the amount of work conducted by the system administrator. The size and complexity of the scenarios were varied between each scenario to give an indication of the trends for the metrics measured. As the collected data was analyzed, it became clear that administration of an environment had indeed been simplified and more automated with the approach developed in this project.

In the bigger picture, this thesis has shown that the viability of increased automation in z/VM is considerable. It has also given an estimation of the extent of the possible gains that can be achieved through automation. Making z/VM administration more scalable is an important endeavor, and this project has displayed significant improvement in that area.

With some further work outlined in this report, it is believed that the plugin can be implemented in a production environment.

Looking back at the problem statement of the thesis, it is believed to have been answered in a satisfactory manner. Although MLN does not currently offer *all* of its original functionality for z/VM, it is not predicted to present significant problems to add the remaining functionality.

6.1 Future Work

6.1.1 Improve MLN z/VM plugin

Although the MLN z/VM plugin works in its current state, it is dependent on completely correct user input and a problem free execution of the code. In the future, checks and failover code should be added to bring it up to the robustness level that MLN currently has. Also, the flexibility must be improved to handle more situations like multiple template formats.

6.1.2 DirMaint independence

DirMaint is currently the only non-free IBM product used in the architecture. It would be an interesting project to remove it from the architecture by introducing a free alternative or designing an alternative storage management system from scratch. To create a system that could rival DirMaint in security and speed will probably be quite a challenge and a large scale undertaking.

6.1.3 z/VM options in MLN z/VM plugin

A future project could revolve around introducing z/VM specific options and features in the plugin. The plugin would still retain its original purpose, to be used by users without z/VM knowledge, but would offer more to users with z/VM experience. This could be a set of attributes giving more direct z/VM control. Examples are support for more networking technologies and linking to minidisks.

6.1.4 MLN redesign

This project has been a big step in making MLN technology independent. MLN now has much better capabilities for using plugins for the Virtualization platform code. There are however still a lot more that can be done. A complete reworking of the current MLN base code could move the User-Mode Linux and Xen specific code to separate plugins and design MLN to use plugins for all platform specific code. By making MLN completely technology independent and plugin based, it would make it possible to add new virtualization technologies without changing the MLN base code.

Bibliography

[1] XenSource. Xen. http://www.xen.org/.

[2] Jeff Dike. User-mode linux. http://user-mode-linux.sourceforge.net/.

[3] Lydia Parziale, Edi Lopes Alves, Eli M. Dow, Klaus Egeler, Jason J. Herne, Clive Jordan, Eravimangalath P. Naveen, Manoj S. Pattabhiraman, and Kyle Smith. *Introduction to the New Mainframe: z/VM Basics*. IBM, 2007.

[4] Gregory Geiselhart, Malcolm Beattie, Vic Cross, Michael Donovan, Aaron Kirby, Lutz Kühner, Julie Murhpy, and Michael Weisbach. *Linux on IBM eServer zSeries and S/390: Large Scale Linux Deployment*. IBM, 2002.

[5] *z/VM CMS Planning and Administration Version 4 Release 3.0*. IBM, 2002.

[6] Kyrre M. Begnum. Managing large networks of virtual machines. *Proceedings of the Twentieth System Administration Conference (LISA XX) (USENIX Association: Berkeley, CA)*, 2006.

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

[8] Paul Mattes. x2370. http://www.geocities.com/siliconvalley/peaks/7814/index.html.

[9] *z/VM CP Commands and Utilities Reference Version 5 Release 2*. IBM, 2005.

[10] *z/VM CP Planning and Administration Version 5 Release 1*. IBM, 2007.

[11] *z/VM Directory Maintenance Facility Commands Reference Version 5 Release 3*. IBM, 2007.

[12] *z/VM Directory Maintenance Facility Tailoring and Administration Guide Version 5 Release 3*. IBM, 2007.

[13] *Program Directory for IBM z/VM Directory Maintenance Facility Feature function level 510*. IBM, 2005.

[14] Gregory Geiselhart, Robert Brenneman, Eli Dow, Klaus Egeler, Torsten Gutenberger, Bruce Hayden, and Livio Sousa. *Linux for IBM System z9 and IBM zSeries*. IBM, 2006.

[15] *z/VM Getting Started with Linux on System z9 and zSeries Version 5 Release 2*. IBM, 2005.

Appendices

Appendix A

zVM.pl

```
# The z/VM guest name of the PROP Service VM
$mlnprop = "mlnprop";
# The virtual device number used by the admin penguin to link target
    penguins 0100 mdisk
$dasdVirtDevNum = 1337;
# The first virtual device number used to create network interface cards
$qethVirtDev = 700;
# How much to increment $qethVirtDev with for each nic
$qethVirtDevInc = 10;
# The systems outgoing vswitch. used by gateways and NICs with no defined
    switch
$mlnvsw = LOCALNET;
# Cylinder size in bytes
$cylsize = 849960;

sub zVM_postParse {
  out( "zVM_postParse called\n" );

  # we need to check all the hosts in the same method.

  # we use 'getHosts()' instead of 'keys getHash("/host")'
  # because it has better performance and we
  # dont need the entire hash anyway

  # checks if this is a IBM mainframe environment by looking at the cpu
  if (`cat /proc/cpuinfo | grep IBM/S390`)
  {
    my $host;
    foreach $host ( getHosts() ){
      setScalar("/host/$host/zVM","1");
      if (getScalar("/host/$host/xen") == "1") {
        setScalar("/host/$host/xen","0");
      }
      if (getScalar("/host/$host/uml") == "1") {
        setScalar("/host/$host/uml","0");
      }
      my $zname = genzname($host, getScalar("/global/project"));
      setScalar("/host/$host/zname", $zname);
    }
  }

  out( "zVM_postParse finished\n" );
}
```

```
42
43 sub execute {
44   if ( getScalar("/global/dryrun") ) {
45     out( "  Dryrun: " . $_[0] . "\n" );
46   }
47   else {
48     system( $_[0] );
49   }
50 }
51
52 sub genzname {
53   my $name = $_[0];
54   my $proj = $_[1];
55
56   $name =~ s/^.*(\S\S\S\S)$/$1/;
57   $proj =~ s/^.*(\S\S\S\S)$/$1/;
58
59   return uc($name) . uc($proj);
60 }
61
62 sub zVM_createFilesystem {
63   out( "zVM_createFilesystem called\n" );
64
65   my $hostname = $_[0];
66   my $zname = genzname($hostname,$PROJECT);
67
68   out( "Generating z/VM guest name: $hostname $PROJECT -> $zname\n" );
69
70   my $memsize = getScalar("/host/$hostname/memory");
71   my $template = getScalar("/host/$hostname/template");
72   my $maxmemsize = $memsize;
73   my $disksize = getScalar("/host/$hostname/size");
74   $disksize = $DEFAULTS{FILESYSTEM_SIZE} unless $disksize;
75
76   my %BLOCK_UNITS = (
77       'b'  => 512,
78       'kB' => 1000,
79       'KB' => 1000,
80       'K', => 1024,
81       'MB' => 1000 * 1000,
82       'M'  => 1024 * 1024,
83       'GB' => 1000 * 1000 * 1000,
84       'G'  => 1024 * 1024 * 1024,
85   );
86
87   my $unit = $disksize;
88   $unit =~ tr/0-9//d;
89   $disksize =~ tr/A-Za-z//d;
90   my $cylinders = ceil(($disksize * $BLOCK_UNITS{$unit})/$cylsize);
91
92   out( "Disksize $disksize$unit -> $cylinders cylinders\n" );
93
94   ## DirMaint commands
95   ## Although the ideal solution would be to have the admin linux guest
         issue the dirmaint
96   ## commands directly, it is (as far as I know) difficult at best,
         imposibe at worst.
97   ## Since dirmaint commands are CMS commands, they cannot be issued
         directly to CP.
98   ## And because Linux and CMS cannot run simultaniously in a guest, the
         commands must
99   ## go through a guest running CMS. In this case the PROP.
```

```
100   ## For convenience, and in case a solution to the problem is found, the
          apropriate
101   ## dirmaint commands are provided as comments.
102
103   # creates the guest from a prototype template
104   $linuxPrototype = mlnlinux;
105   # Since the admin linux guest does not initially have any rights in
          dirmaint, the creation
106   # of a new guest must always go through PROP.
107 #  execute("vmcp dirmaint add $zname like $linuxPrototype pw lbyonly");
108   out( "$mlnprop -> Create new z/VM guest $zname from prototype
          $linuxPrototype\n" );
109   execute("vmcp smsg $mlnprop dirmadd $zname $linuxPrototype");
110   sleep 5;
111   # This command would give the admin guest authority to directly
          influence the new guest
112   # Since all commands must go through PROP, this command is currently
          unnecessary.
113 #  execute("vmcp smsg $mlnprop dirmauth $zname");
114   # sets the available memory (storage) and max memory for the guest
115 #  execute("vmcp dirmaint for $zname maxstore $maxmemsize");
116 #  execute("vmcp dirmaint for $zname storage $memsize");
117   out( "$mlnprop -> Give $zname $memsize of memory\n" );
118   execute("vmcp smsg $mlnprop dirmstorage $zname $memsize");
119   sleep 4;
120   # creates a new mdisk of size $disksize  for the guest
121 #  execute("vmcp dirmaint for $zname amdisk 0100 x autog $disksize mln mr
        pws all all all");
122   out( "$mlnprop -> Create $cylinders cylinder disk for $zname\n" );
123   execute("vmcp smsg $mlnprop dirmamdisk $zname 0100 x autog $cylinders
          mln mr pws all all all");
124   sleep 5;
125   # creates network interface cards for the new guest
126   my $qethVirtDevLocal = $qethVirtDev;
127   my %network = getHash("/host/$hostname/network");
128   my $vswitch;
129   foreach my $if ( keys %network ) {
130     out( "$mlnprop -> Create network interface card for $zname\n" );
131     if ( getScalar("/host/$hostname/network/$if/switch") ) {
132       $vswitch = getScalar("/host/$hostname/network/$if/switch");
133 #       execute("vmcp dirmaint for $zname nicdef $qethVirtDevLocal type
        qdio lan system $vswitch");
134       execute("vmcp smsg $mlnprop dirmnicdef $zname $qethVirtDevLocal type
              qdio lan system $vswitch");
135     }
136     else {
137 #       execute("vmcp dirmaint for $zname nicdef $qethVirtDevLocal type
        qdio lan system $mlnvsw");
138       execute("vmcp smsg $mlnprop dirmnicdef $zname $qethVirtDevLocal type
              qdio lan system $mlnvsw");
139       # convergant, so ok to run several times
140       execute("vmcp smsg $mlnprop setvswitch $mlnvsw grant $zname");
141     }
142     $qethVirtDevLocal += $qethVirtDevInc;
143   }
144
145   # Link new 0100 disk to admin penguin
146   execute("vmcp link $zname 0100 $dasdVirtDevNum mr");
147   sleep 3;
148   # FlashCopy can be done at this point:
149 #  execute("vmcp flashcopy $templateDisk 0 end to $dasdVirtDevNum 0 end");
150   # Activate disk
```

```
151   execute("chccwdev -e $dasdVirtDevNum");
152   # Copy template to disk
153   # As an alternative to FlashCopy, dd can be done at this point
154   # If dd is to be used, the new disk must first be formated for linux
155   my $lsdasd_line = `lsdasd 0.0.$dasdVirtDevNum`;
156   my @lsdasd = split /\s+/, $lsdasd_line;
157   my $dasdDev = $lsdasd[6];
158
159   out( "Formating /dev/$dasdDev\n" );
160   execute("dasdfmt -b 4096 -y -f /dev/$dasdDev");
161
162   out( "Unzipping and applying $template to /dev/$dasdDev\n" );
163   out( "Please wait. This may take a while...\n" );
164   execute("dd if=$template | gunzip | dd of=/dev/$dasdDev");
165
166   execute("chccwdev -d $dasdVirtDevNum");
167   execute("vmcp detach $dasdVirtDevNum");
168
169   out( "zVM_createFilesystem finished\n" );
170   return 1;
171 }
172
173 sub zVM_mountFilesystem {
174   out( "zVM_mountFilesystem called\n" );
175
176   my $hostname = $_[0];
177   my $zname = genzname($hostname,$PROJECT);
178
179   execute("vmcp link $zname 0100 $dasdVirtDevNum mr");
180   execute("chccwdev -e $dasdVirtDevNum");
181
182   # the dasd device name as it appears in /dev/
183   my $dasdDev;
184   my $lsdasd_line = `lsdasd 0.0.$dasdVirtDevNum`;
185   my @lsdasd = split /\s+/, $lsdasd_line;
186   $dasdDev = $lsdasd[6];
187   # because there is only one partition in the root filesystem, it will be
          identified as /dev/dasdX1
188   $dasdDev .= "1";
189   out( "mounting /dev/$dasdDev on $MOUNTDIR\n" );
190   execute("mount /dev/$dasdDev $MOUNTDIR");
191   execute("df -H");
192
193   out( "zVM_mountFilesystem finished\n" );
194   return 1;
195 }
196
197 sub zVM_removeHost {
198   out( "zVM_removeHost called\n" );
199
200   my $hostname = $_[0];
201   my $project = $_[1];
202   my $zname = genzname($hostname,$project);
203   my $checkIfUp = zVM_checkIfUp($hostname,$project);
204   if ( $checkIfUp == -1 ) {
205     execute("vmcp smsg $mlnprop setvswitch $mlnvsw revoke $zname");
206     out( "$mlnprop -> Revoke access to $mlnvsw for $zname\n" );
207     execute("vmcp smsg $mlnprop dirmpurge $zname");
208     out( "$mlnprop -> PURGE $zname command issued\n" );
209   }
210   elsif ( $checkIfUp == 1 ) {
```

```
211     out( "Warning: $zname appears to be running. Please stop project
            before removing it\n" );
212   }
213
214   out( "zVM_removeHost finished\n" );
215   return 1;
216 }
217
218 sub zVM_configure {
219   out( "zVM_configure called\n" );
220
221   my $host;
222   foreach $host ( getHosts() ) {
223     my $qethVirtDevLocal = $qethVirtDev;
224     my %network = getHash("/host/$host/network");
225     foreach my $if ( keys %network ){
226       my $ipaddr = getScalar("/host/$host/network/$if/address");
227       my $netmask = getScalar("/host/$host/network/$if/netmask");
228       my $broadcast = getScalar("/host/$host/network/$if/broadcast");
229       print "NIC ".$if." -> ".$ipaddr."\n";
230       print "NIC ".$if." -> ".$qethVirtDevLocal."\n";
231
232       execute ("mkdir -p $MOUNTDIR/etc/sysconfig/network/");
233       out( "Writing $MOUNTDIR/etc/sysconfig/network/ifcfg-qeth-bus-ccw
              -0.0.0.$qethVirtDevLocal on $host\n" );
234       open( IFCFG, ">$MOUNTDIR/etc/sysconfig/network/ifcfg-qeth-bus-ccw
              -0.0.0".$qethVirtDevLocal )
235         or warn "Failed to open ifcfg-qeth-bus-ccw-0.0.0".
              $qethVirtDevLocal;
236       print IFCFG "BOOTPROTO=\"static\"\n";
237       print IFCFG "UNIQUE=\"\"\n";
238       print IFCFG "STARTMODE=\"onboot\"\n";
239       print IFCFG "IPADDR=\"$ipaddr\"\n";
240       print IFCFG "NETMASK=\"$netmask\"\n";
241       print IFCFG "BROADCAST=\"$broadcast\"\n";
242       # because the current outgoing vswitch is layer 3
243       if ( getScalar("/host/$host/network/$if/switch") ) {
244         print IFCFG "ARP=\"yes\"\n";
245       }
246       print IFCFG "_nm_name='qeth-bus-ccw-0.0.0".$qethVirtDevLocal."'\n";
247       close(IFCFG);
248
249       execute ("mkdir -p $MOUNTDIR/etc/sysconfig/hardware/");
250       out( "Writing $MOUNTDIR/etc/sysconfig/hardware/hwcfg-qeth-bus-ccw
              -0.0.0.$qethVirtDevLocal on $host\n" );
251       open( HWCFG, ">$MOUNTDIR/etc/sysconfig/hardware/hwcfg-qeth-bus-ccw
              -0.0.0".$qethVirtDevLocal )
252         or warn "Failed to open hwcfg-qeth-bus-ccw-0.0.0".
              $qethVirtDevLocal;
253       print HWCFG "STARTMODE=\"auto\"\n";
254       print HWCFG "MODULE=\"qeth\"\n";
255       print HWCFG "MODULE_OPTIONS=\"\"\n";
256       print HWCFG "MODULE_UNLOAD=\"yes\"\n";
257       print HWCFG "SCRIPTUP=\"hwup-ccw\"\n";
258       print HWCFG "SCRIPTUP_ccw=\"hwup-ccw\"\n";
259       print HWCFG "SCRIPTUP_ccwgroup=\"hwup-qeth\"\n";
260       print HWCFG "SCRIPTDOWN=\"hwdown-ccw\"\n";
261       print HWCFG "CCW_CHAN_IDS=\"0.0.0".$qethVirtDevLocal." 0.0.0".(
              $qethVirtDevLocal+1)." 0.0.0".($qethVirtDevLocal+2)."\"\n";
262       print HWCFG "CCW_CHAN_NUM=\"3\"\n";
263       # because the current outgoing vswitch is layer 3
264       if ( getScalar("/host/$host/network/$if/switch") ) {
```

```
265         print HWCFG "QETH_LAYER2_SUPPORT=\"1\"\n";
266       }
267       else {
268         print HWCFG "QETH_LAYER2_SUPPORT=\"0\"\n";
269       }
270       close(HWCFG);
271
272       my $gateway_address = getScalar("/host/$host/network/$if/gateway");
273       if ( $gateway_address ) {
274         out( "Writing gateway $gateway_address to $MOUNTDIR/etc/sysconfig/
              network/routes\n" );
275         open( GW, ">$MOUNTDIR/etc/sysconfig/network/routes" )
276           or warn "Failed to open routes";
277         print GW "default ".$gateway_address." - -\n";
278         close(GW);
279       }
280
281       out("Writing hostname $host to $MOUNTDIR/etc/HOSTNAME\n");
282       open( HOST, ">$MOUNTDIR/etc/HOSTNAME" )
283         or warn "Failed to open HOSTNAME";
284       print HOST $host."\n";
285       close(HOST);
286
287       $qethVirtDevLocal += $qethVirtDevInc;
288     }
289   }
290
291   out( "zVM_configure finished\n" );
292 }
293
294 sub zVM_unmountFilesystem {
295   out( "zVM_unmountFilesystem called\n" );
296
297   out( "Unmounting $MOUNTDIR\n" );
298   execute("umount $MOUNTDIR");
299   execute("chccwdev -d $dasdVirtDevNum");
300   execute("vmcp detach $dasdVirtDevNum");
301
302   out( "zVM_unmountFilesystem finished\n" );
303   return 1;
304 }
305
306 sub zVM_createStartStopScripts {
307   out( "zVM_createStartStopScripts called\n" );
308
309   ## create start and stop scripts for each vm and switch
310   # create host startscript
311   my $hostname = $_[0];
312   my $boot_order = getScalar("/host/$hostname/boot_order");
313   $boot_order = $DEFAULTS{'BOOT_ORDER'} unless $boot_order;
314   my $zname = genzname($hostname,$PROJECT);
315
316   # remove existing start and stop scripts
317   my $old_order = getScalar( "/host/$hostname/boot_order",$OLD_DATA_ROOT )
          ;
318   if ( $old_order and $old_order != $boot_order ) {
319     verbose("Removing old start and stop script\n");
320     execute("$shell{'RM'} $PROJECT_PATH/$PROJECT/start_${old_order}
          _$hostname.sh");
321     execute("$shell{'RM'} $PROJECT_PATH/$PROJECT/stop_${old_order}
          _$hostname.sh");
322   }
```

```
323
324   # writing start script
325   out( "Writing $PROJECT_PATH/$PROJECT/start_${boot_order}_$hostname.sh\n"
          );
326   open( HOSTSTART,">$PROJECT_PATH/$PROJECT/start_${boot_order}_$hostname.
          sh" )
327     or warn "failed to open start_$hostname\n";
328   print HOSTSTART "#! /bin/sh\n";
329   print HOSTSTART "echo " if ( getScalar("/global/dryrun") );
330   print HOSTSTART "vmcp smsg $mlnprop xautolog $zname\n";
331   close (HOSTSTART);
332   system("chmod 755 $PROJECT_PATH/$PROJECT/start_${boot_order}_$hostname.
          sh");
333
334   # writing stop script
335   out( "Writing $PROJECT_PATH/$PROJECT/stop_${boot_order}_$hostname.sh\n"
          );
336   open( STOP, ">$PROJECT_PATH/$PROJECT/stop_${boot_order}_$hostname.sh" )
337     or warn "Failed to open $STOP_SCRIPT\n";
338   print STOP "#!/bin/sh\n";
339   print STOP "if \[ \$1 \]; then\n";
340   print STOP "echo " if ( getScalar("/global/dryrun") );
341   print STOP "vmcp smsg $mlnprop force $zname\n";
342   print STOP "exit\n";
343   print STOP "else\n";
344   print STOP "echo " if ( getScalar("/global/dryrun") );
345   print STOP "vmcp smsg $mlnprop signalshutdown $zname\n";
346   print STOP "exit\n";
347   print STOP "fi\n";
348   close(STOP);
349   system("chmod 755 $PROJECT_PATH/$PROJECT/stop_${boot_order}_$hostname.sh
          ");
350
351   out( "zVM_createStartStopScripts finished\n" );
352 }
353
354 sub zVM_checkIfUp {
355   out( "zVM_checkIfUp called\n" );
356
357   ## checks if a given z/VM guest is running
358   my $hostname = $_[0];
359   my $project = $_[1];
360   # this is needed for all getScalar operations
361   my $root = $_[2];
362
363   if (getScalar("/host/$hostname/zVM",$root) == "1") {
364     my $zname = genzname($hostname,$project);
365
366     my $guestquery = `vmcp query users $zname  2>/dev/null`;
367     my @gqarray = split( / /, $guestquery);
368
369     if ( $gqarray[0] eq $zname ) {
370       out( "zVM_checkIfUp finished (returning 1)\n" );
371       return 1;
372     }
373     else {
374       out( "zVM_checkIfUp finished (returning -1)\n" );
375       return -1;
376     }
377   }
378   else {
379     out( "zVM_checkIfUp finished (returning 0)\n" );
```

```
380     return 0;
381   }
382 }
383
384 sub zVM_checkIfSwitchIsUp {
385   out( "zVM_checkIfSwitchIsUp called\n" );
386
387   ## checks if a given z/VM VSWITCH is running
388   my $switchname = $_[0];
389   my $project = $_[1];
390   my $zname = genzname($switchname,$project);
391
392   my $vswitchquery =  `vmcp query vswitch $zname  2>/dev/null`;
393   my @vswqarray = split( / /, $vswitchquery);
394
395   if ($vswqarray[0].$vswqarray[1].$vswqarray[2] eq "VSWITCH SYSTEM".$zname
          ) {
396     out( "zVM_checkIfSwitchIsUp finished (returning 1)\n" );
397     return 1;
398   }
399   elsif ($vswqarray[1].$vswqarray[2].$vswqarray[3] eq "VSWITCH SYSTEM".
         $zname ) {
400     out( "zVM_checkIfSwitchIsUp finished (returning -1)\n" );
401     return -1;
402   }
403   else {
404     out( "zVM_checkIfSwitchIsUp finished (returning 0)\n" );
405     return 0;
406   }
407 }
408
409 sub zVM_configureSwitch {
410   out( "zVM_configureSwitch called\n" );
411
412   my $name = $_[0];
413   my $zname = genzname($name,$PROJECT);
414
415   printBlock($DATA_ROOT);
416
417   out( "Writing $PROJECT_PATH/$PROJECT/start_$name.sh\n" );
418   open( START, ">$PROJECT_PATH/$PROJECT/start_$name.sh" )
419     or warn "Failed to open $PROJECT_PATH/$PROJECT/start_$name.sh\n";
420   print START "#!/bin/sh\n";
421   print START "echo " if ( getScalar("/global/dryrun") );
422   print START "vmcp smsg $mlnprop definevswitch $zname\n";
423
424   my $host;
425   foreach $host ( getHosts() ) {
426     my $zhost = getScalar("/host/$host/zname");
427     my %network = getHash("/host/$host/network");
428     foreach my $if ( keys %network ){
429       if ( getScalar("/host/$host/network/$if/switch") eq $name ) {
430         out( "Assigning $host.$if -> $name\n" );
431         print START "echo " if ( getScalar("/global/dryrun") );
432         print START "vmcp smsg $mlnprop setvswitch $zname grant $zhost\n";
433       }
434     }
435   }
436
437   close(START);
438
439   out( "Writing $PROJECT_PATH/$PROJECT/stop_$name.sh\n" );
```

```
440   open( STOP, ">$PROJECT_PATH/$PROJECT/stop_$name.sh" )
441     or warn "Failed to open $PROJECT_PATH/$PROJECT/stop_$name.sh\n";
442   print STOP "#!/bin/sh \n\n";
443   print START "echo " if ( getScalar("/global/dryrun") );
444   print STOP "vmcp smsg $mlnprop detachvswitch $zname\n";
445   close(STOP);
446
447   out( "zVM_configureSwitch finished\n" );
448 }
449
450 1;
```

Appendix B

PROP RTABLE

```
1  ===== * * * Top of File * * *
2  ===== LGLOPR MAINT HIOVM2
3  ===== TEXTSYM / $ ^
4  ===== LOGGING ALL
5  ===== ROUTE
6  ===== *
7  ===== *COMMANDS
8  ===== *
9  ===== /XAUTOLOG /                1   9  4 LINUX1           PENGUINS
       XAUTOLOG
10 ===== /SIGNALSHUTDOWN /          1  15  4 LINUX1           PENGUINS
       SIGSHUTD
11 ===== /FORCE /                   1   6  4 LINUX1           PENGUINS
       FORCE
12 ===== /DEFINEVSWITCH /           1  14  4 LINUX1           PENGUINS
       DEFVSW
13 ===== /SETVSWITCH /              1  11  4 LINUX1           PENGUINS
       SETVSW
14 ===== /DETACHVSWITCH /           1  14  4 LINUX1           PENGUINS
       DETVSW
15 ===== /DIRMADD /                 1   8  4 LINUX1           PENGUINS
       DIRMADD
16 ===== /DIRMAUTH /                1   9  4 LINUX1           PENGUINS
       DIRMAUTH
17 ===== /DIRMSTORAGE /             1  12  4 LINUX1           PENGUINS
       DIRMSTOR
18 ===== /DIRMAMDISK /              1  11  4 LINUX1           PENGUINS
       DIRMAMD
19 ===== /DIRMNICDEF /              1  11  4 LINUX1           PENGUINS
       DIRMNICD
20 ===== /DIRMPURGE /               1  10  4 LINUX1           PENGUINS
       DIRMPRG
21 ===== * * * End of File * * *
```

Appendix C

PENGUINS EXEC

```
===== * * * Top of File * * *
===== /* PROP ACTION SCRIPT USED TO VALIDATE COMMANDS AND SO ON */
=====
===== conffile="RESGUEST CONF"
=====
===== parse upper arg ruser rnode lglopr msgcode puser pnode netid rtable
===== pull msg
===== pull action
=====
===== /*
===== * checks if the username is "legal"
===== */
===== parse var msg . user restofmsg
===== okchars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
===== okchars = okchars !! "0123456789_-$"
===== if verify(user, okchars, "Nomatch") ! length(user)=0 ! length(user)
    >8
===== then do
=====   say "PENGUINS: Syntax error in requested username:" user
=====   exit 1
===== end
=====
===== /*
===== * do not allow a username which is listed in RESGUEST CONF
===== */
===== found = 0
===== parse value stream(conffile,'c','open read') with ok fh
===== if ok == "ERROR:" then do
=====   say "PENGUINS: Error opening config file" conffile ":" fh
=====   exit 1
===== end
=====
===== do while found == 0 & lines(fh) > 0
=====   parse value linein(fh) with cuser .
=====   if translate(cuser) == translate(user) then found = 1
===== end
===== ok = stream(fh,'c','close')
=====
===== if found == 1 then do
=====   say "PENGUINS: Target guest listed as invalid target:" user
=====   exit 1
===== end
=====
```

```
===== /*
===== * the actuall actions to be taken as a result of the incomming msg
===== */
===== select
=====   when action = "XAUTOLOG" then
=====     address 'CMS' "XAUTOLOG" user
=====   when action = "SIGSHUTD" then
=====     address 'CMS' "SIGNAL SHUTDOWN" user
=====   when action = "FORCE" then
=====     address 'CMS' "FORCE" user
=====   when action = "DEFVSW" then
=====     address command CP "DEFINE VSWITCH" user "ETHERNET"
=====   when action = "SETVSW" then
=====     address command CP "SET VSWITCH" user restofmsg
=====   when action = "DETVSW" then
=====     address command CP "DETACH VSWITCH" user
=====   when action = "DIRMADD" then
=====     address 'CMS' "DIRM ADD" user "LIKE" restofmsg "PW LBYONLY"
=====   when action = "DIRMAUTH" then do
=====   address 'CMS' "DIRM FOR" user "AUTHFOR LINUX1 CMDL 140A CMDS
    ADGHMOPSZ"
=====   address 'CMS' "DIRM FOR" user "AUTHFOR LINUX1 CMDL 150A CMDS
    ADGHMOPSZ"
=====   end
=====   when action = "DIRMSTOR" then do
=====     address 'CMS' "DIRM FOR" user "MAXSTORE" restofmsg
=====     address 'CMS' "DIRM FOR" user "STORAGE" restofmsg
=====   end
=====   when action = "DIRMAMD" then
=====     address 'CMS' "DIRM FOR" user "AMDISK" restofmsg
=====   when action = "DIRMNICD" then
=====     address 'CMS' "DIRM FOR" user "NICDEF" restofmsg
=====   when action = "DIRMPRG" then
=====     address 'CMS' "DIRM FOR" user "PURGE NOCLEAN"
=====   otherwise do
=====     say "PENGUINS: Unknown action:" action
=====     exit 1
=====   end
===== end
===== exit 0
===== * * * End of File * * *
```

Appendix D

RESGUEST CONF

```
1  ===== * * * Top of File * * *
2  ===== MAINT
3  ===== DTCVSW1
4  ===== DTCVSW2
5  ===== OPERSYMP
6  ===== DISKACNT
7  ===== EREP
8  ===== OPERATOR
9  ===== MLNPROP
10 ===== LINUX1
11 ===== DIRMAINT
12 ===== DATAMOVE
13 ===== 5VMDIR10
14 ===== * * * End of File * * *
```

Appendix E

EXTENT CONTROL

```
1  ===== * * * Top of File * * *
2  ===== *
       ********************************************************************
3  ===== * CopyRight Notice
4  ===== *  LICENSED MATERIALS - PROGRAM PROPERTY OF IBM.
5  ===== *  RESTRICTED MATERIALS OF IBM.
6  ===== *  5741-A05  (C) COPYRIGHT IBM CORPORATION 1979, 2004.
7  ===== *  All rights reserved.
8  ===== *  US Government Users Restricted Rights -
9  ===== *  Use, duplication, or disclosure restricted by GSA ADP
10 ===== *  schedule contract with IBM Corporation.
11 ===== *  Status: 510
12 ===== *  PITS:   @Z----ID
13 ===== *  APAR:   @VA-----
14 ===== *
15 ===== * Purpose:   Default Extent Control file.
16 ===== *
17 ===== * Change Activity: (IBM)
18 ===== * @V715DSR - New for DirMaint Version 1 Release 5 Mod 0.
19 ===== * @VA61019 - Support for Multiprise 2000 variable size DASD.
20 ===== * @VA61648 - Support 1084 cylinder emulated 3390 DASD.
21 ===== * @VRA8FHA - Remove AUTOBLK and DEFAULTS sections.
22 ===== *            Note: Valid TYPE values for REGIONS are now defined
23 ===== *            in the DEFAULTS DATADVH file.
24 ===== *            Overrides may still be included here.
25 ===== * @VRFCWRS - Re-structure the REGIONS section header to support
26 ===== *            10 digits for RegStart & RegEnd.
27 ===== * ... (reserved for future change activity) ...
28 ===== * ... (reserved for future change activity) ...
29 ===== * ... (reserved for future change activity) ...
30 ===== * ... (reserved for future change activity) ...
31 ===== * ... (reserved for future change activity) ...
32 ===== * ... (reserved for future change activity) ...
33 ===== * ... (reserved for future change activity) ...
34 ===== * Change Activity: (Local)
35 ===== * None (yet).
36 ===== *
37 ===== *
       ********************************************************************
38 ===== :REGIONS.
39 =====   *RegionId  VolSer    RegStart      RegEnd  Dev-Type  Comments
40 =====    LINUX01   USER03           1         END  3390-03
41 =====    LINUX02   USER02           1         END  3390-03
```

```
=====    LINUX03   520W01        2855        3234  3390-03
===== *  LINUX03   520W02        1478        1527  3390-03
===== *  LINUX04   520W02        1828         END  3390-03
===== :END.
===== :GROUPS.
=====   *GroupName RegionList
=====  MLN       LINUX01  LINUX02  LINUX03
===== :END.
===== :EXCLUDE.
===== * USERID ADDRESS
===== MAINT 012*
===== SYSDUMP1 012*
===== $FULLPK$ *
===== :END.
===== :AUTOBLOCK.
=====   * IBM supplied defaults are contained in the AUTOBLK DATADVH file.
=====   * The following are customer overrides and supplements.
=====   *
=====   *DASDType BlockSize Blocks/Unit Alloc_Unit Architecture
===== :END.
===== :DEFAULTS.
=====   * IBM supplied defaults are contained in the DEFAULTS DATADVH file
    .
=====   * The following are customer overrides and supplements.
=====   *
=====   *DASDType Max-Size
=====    3390-03  3338
===== :END.
===== * * * End of File * * *
```

Appendix F

AUTHFOR CONTROL

```
1 ===== * * * Top of File * * *
2 ===== ALL MAINT    *        140A ADGHOPSMZ
3 ===== ALL MAINT    *        150A ADGHOPSMZ
4 ===== ALL MLNPROP  *        140A ADGHOPSMZ
5 ===== ALL MLNPROP  *        150A ADGHOPSMZ
6 ===== * * * End of File * * *
```

Appendix G

Scenario III VSWITCH commands

```
DEFINE VSWITCH SW11RIO3 ETHERNET
DEFINE VSWITCH SW12RIO3 ETHERNET
DEFINE VSWITCH SW13RIO3 ETHERNET
DEFINE VSWITCH SW14RIO3 ETHERNET
DEFINE VSWITCH SW21RIO3 ETHERNET
DEFINE VSWITCH SW22RIO3 ETHERNET
DEFINE VSWITCH SW31RIO3 ETHERNET
DEFINE VSWITCH SW32RIO3 ETHERNET
SET VSWITCH GATEWAY GRANT LB1RIO3
SET VSWITCH GATEWAY GRANT LB2RIO3
SET VSWITCH GATEWAY GRANT LB3RIO3
SET VSWITCH GATEWAY GRANT LB4RIO3
SET VSWITCH SW11RIO3 GRANT LB1RIO3
SET VSWITCH SW11RIO3 GRANT WS1RIO3
SET VSWITCH SW11RIO3 GRANT WS2RIO3
SET VSWITCH SW11RIO3 GRANT WS3RIO3
SET VSWITCH SW11RIO3 GRANT WS4RIO3
SET VSWITCH SW11RIO3 GRANT WS5RIO3
SET VSWITCH SW11RIO3 GRANT WS6RIO3
SET VSWITCH SW11RIO3 GRANT WS7RIO3
SET VSWITCH SW11RIO3 GRANT WS8RIO3
SET VSWITCH SW12RIO3 GRANT LB2RIO3
SET VSWITCH SW12RIO3 GRANT WS1RIO3
SET VSWITCH SW12RIO3 GRANT WS2RIO3
SET VSWITCH SW12RIO3 GRANT WS3RIO3
SET VSWITCH SW12RIO3 GRANT WS4RIO3
SET VSWITCH SW12RIO3 GRANT WS5RIO3
SET VSWITCH SW12RIO3 GRANT WS6RIO3
```

```
SET VSWITCH SW12RIO3 GRANT WS7RIO3
SET VSWITCH SW12RIO3 GRANT WS8RIO3
SET VSWITCH SW13RIO3 GRANT LB3RIO3
SET VSWITCH SW13RIO3 GRANT WS1RIO3
SET VSWITCH SW13RIO3 GRANT WS2RIO3
SET VSWITCH SW13RIO3 GRANT WS3RIO3
SET VSWITCH SW13RIO3 GRANT WS4RIO3
SET VSWITCH SW13RIO3 GRANT WS5RIO3
SET VSWITCH SW13RIO3 GRANT WS6RIO3
SET VSWITCH SW13RIO3 GRANT WS7RIO3
SET VSWITCH SW13RIO3 GRANT WS8RIO3
SET VSWITCH SW14RIO3 GRANT LB4RIO3
SET VSWITCH SW14RIO3 GRANT WS1RIO3
SET VSWITCH SW14RIO3 GRANT WS2RIO3
SET VSWITCH SW14RIO3 GRANT WS3RIO3
SET VSWITCH SW14RIO3 GRANT WS4RIO3
SET VSWITCH SW14RIO3 GRANT WS5RIO3
SET VSWITCH SW14RIO3 GRANT WS6RIO3
SET VSWITCH SW14RIO3 GRANT WS7RIO3
SET VSWITCH SW14RIO3 GRANT WS8RIO3
SET VSWITCH SW21RIO3 GRANT WS1RIO3
SET VSWITCH SW21RIO3 GRANT WS2RIO3
SET VSWITCH SW21RIO3 GRANT WS3RIO3
SET VSWITCH SW21RIO3 GRANT WS4RIO3
SET VSWITCH SW21RIO3 GRANT LB5RIO3
SET VSWITCH SW22RIO3 GRANT WS5RIO3
SET VSWITCH SW22RIO3 GRANT WS6RIO3
SET VSWITCH SW22RIO3 GRANT WS7RIO3
SET VSWITCH SW22RIO3 GRANT WS8RIO3
SET VSWITCH SW22RIO3 GRANT LB6RIO3
SET VSWITCH SW31RIO3 GRANT LB5RIO3
SET VSWITCH SW31RIO3 GRANT QLS1RIO3
SET VSWITCH SW31RIO3 GRANT QLS2RIO3
SET VSWITCH SW31RIO3 GRANT QLS3RIO3
SET VSWITCH SW31RIO3 GRANT QLS4RIO3
SET VSWITCH SW32RIO3 GRANT LB6RIO3
SET VSWITCH SW32RIO3 GRANT QLS1RIO3
SET VSWITCH SW32RIO3 GRANT QLS2RIO3
SET VSWITCH SW32RIO3 GRANT QLS3RIO3
SET VSWITCH SW32RIO3 GRANT QLS4RIO3
```

Appendix H

Scenario III USER DIRECT file

```
PROFILE MLNLINUX
        CLASS G
        DATEFORMAT FULLDATE
        IPL CMS PARM AUTOCR
        LOGONBY MAINT
        MACHINE ESA
        OPTION TODENABLE
        CONSOLE 0009 3215 T
        SPOOL 000C 2540 READER *
        SPOOL 000D 2540 PUNCH A
        SPOOL 000E 1403 A
        LINK MAINT 0190 0190 RR
        LINK MAINT 019D 019D RR
        LINK MAINT 019E 019E RR
        LINK TCPMAINT 0592 0592 RR
        LINK MAINT 1000 0191 RR

USER LB1RIO3 512M 512M
        PROFILE MLNLINUX
        NICDEF 700 TYPE QDIO LAN SYSTEM GATEWAY
        NICDEF 710 TYPE QDIO LAN SYSTEM SW11RIO3
        MDISK 0100 3390 0001 1665 DISK01 MR ALL ALL ALL
USER LB2RIO3 512M 512M
        PROFILE MLNLINUX
        NICDEF 700 TYPE QDIO LAN SYSTEM GATEWAY
        NICDEF 710 TYPE QDIO LAN SYSTEM SW12RIO3
        MDISK 0100 3390 1666 3330 DISK01 MR ALL ALL ALL
USER LB3RIO3 512M 512M
        PROFILE MLNLINUX
        NICDEF 700 TYPE QDIO LAN SYSTEM GATEWAY
        NICDEF 710 TYPE QDIO LAN SYSTEM SW13RIO3
        MDISK 0100 3390 0001 1665 DISK02 MR ALL ALL ALL
USER LB4RIO3 512M 512M
        PROFILE MLNLINUX
        NICDEF 700 TYPE QDIO LAN SYSTEM GATEWAY
        NICDEF 710 TYPE QDIO LAN SYSTEM SW14RIO3
        MDISK 0100 3390 1666 3330 DISK02 MR ALL ALL ALL
USER LB5RIO3 512M 512M
        PROFILE MLNLINUX
        NICDEF 700 TYPE QDIO LAN SYSTEM SW21RIO3
        NICDEF 710 TYPE QDIO LAN SYSTEM SW31RIO3
        MDISK 0100 3390 0001 1665 DISK03 MR ALL ALL ALL
USER LB6RIO3 512M 512M
```

```
        PROFILE MLNLINUX
        NICDEF 700 TYPE QDIO LAN SYSTEM SW22RIO3
        NICDEF 710 TYPE QDIO LAN SYSTEM SW32RIO3
        MDISK 0100 3390 1666 3330 DISK03 MR ALL ALL ALL

USER WS1RIO3 1G 1G
        PROFILE MLNLINUX
        NICDEF 700 TYPE QDIO LAN SYSTEM SW11RIO3
        NICDEF 710 TYPE QDIO LAN SYSTEM SW12RIO3
        NICDEF 720 TYPE QDIO LAN SYSTEM SW13RIO3
        NICDEF 730 TYPE QDIO LAN SYSTEM SW14RIO3
        NICDEF 740 TYPE QDIO LAN SYSTEM SW21RIO3
        MDISK 0100 3390 0001 24000 DISK04 MR ALL ALL ALL
USER WS2RIO3 1G 1G
        PROFILE MLNLINUX
        NICDEF 700 TYPE QDIO LAN SYSTEM SW11RIO3
        NICDEF 710 TYPE QDIO LAN SYSTEM SW12RIO3
        NICDEF 720 TYPE QDIO LAN SYSTEM SW13RIO3
        NICDEF 730 TYPE QDIO LAN SYSTEM SW14RIO3
        NICDEF 740 TYPE QDIO LAN SYSTEM SW21RIO3
        MDISK 0100 3390 0001 24000 DISK05 MR ALL ALL ALL
USER WS3RIO3 1G 1G
        PROFILE MLNLINUX
        NICDEF 700 TYPE QDIO LAN SYSTEM SW11RIO3
        NICDEF 710 TYPE QDIO LAN SYSTEM SW12RIO3
        NICDEF 720 TYPE QDIO LAN SYSTEM SW13RIO3
        NICDEF 730 TYPE QDIO LAN SYSTEM SW14RIO3
        NICDEF 740 TYPE QDIO LAN SYSTEM SW21RIO3
        MDISK 0100 3390 0001 24000 DISK06 MR ALL ALL ALL
USER WS4RIO3 1G 1G
        PROFILE MLNLINUX
        NICDEF 700 TYPE QDIO LAN SYSTEM SW11RIO3
        NICDEF 710 TYPE QDIO LAN SYSTEM SW12RIO3
        NICDEF 720 TYPE QDIO LAN SYSTEM SW13RIO3
        NICDEF 730 TYPE QDIO LAN SYSTEM SW14RIO3
        NICDEF 740 TYPE QDIO LAN SYSTEM SW21RIO3
        MDISK 0100 3390 0001 24000 DISK07 MR ALL ALL ALL
USER WS5RIO3 1G 1G
        PROFILE MLNLINUX
        NICDEF 700 TYPE QDIO LAN SYSTEM SW11RIO3
        NICDEF 710 TYPE QDIO LAN SYSTEM SW12RIO3
        NICDEF 720 TYPE QDIO LAN SYSTEM SW13RIO3
        NICDEF 730 TYPE QDIO LAN SYSTEM SW14RIO3
        NICDEF 740 TYPE QDIO LAN SYSTEM SW22RIO3
        MDISK 0100 3390 0001 24000 DISK08 MR ALL ALL ALL
USER WS6RIO3 1G 1G
        PROFILE MLNLINUX
        NICDEF 700 TYPE QDIO LAN SYSTEM SW11RIO3
        NICDEF 710 TYPE QDIO LAN SYSTEM SW12RIO3
        NICDEF 720 TYPE QDIO LAN SYSTEM SW13RIO3
        NICDEF 730 TYPE QDIO LAN SYSTEM SW14RIO3
        NICDEF 740 TYPE QDIO LAN SYSTEM SW22RIO3
        MDISK 0100 3390 0001 24000 DISK09 MR ALL ALL ALL
USER WS7RIO3 1G 1G
        PROFILE MLNLINUX
        NICDEF 700 TYPE QDIO LAN SYSTEM SW11RIO3
        NICDEF 710 TYPE QDIO LAN SYSTEM SW12RIO3
        NICDEF 720 TYPE QDIO LAN SYSTEM SW13RIO3
        NICDEF 730 TYPE QDIO LAN SYSTEM SW14RIO3
        NICDEF 740 TYPE QDIO LAN SYSTEM SW22RIO3
        MDISK 0100 3390 0001 24000 DISK10 MR ALL ALL ALL
USER WS8RIO3 1G 1G
```

```
106         PROFILE MLNLINUX
107         NICDEF 700 TYPE QDIO LAN SYSTEM SW11RIO3
108         NICDEF 710 TYPE QDIO LAN SYSTEM SW12RIO3
109         NICDEF 720 TYPE QDIO LAN SYSTEM SW13RIO3
110         NICDEF 730 TYPE QDIO LAN SYSTEM SW14RIO3
111         NICDEF 740 TYPE QDIO LAN SYSTEM SW22RIO3
112         MDISK 0100 3390 0001 24000 DISK11 MR ALL ALL ALL
113
114 USER QLS1RIO3 1G 1G
115         PROFILE MLNLINUX
116         NICDEF 700 TYPE QDIO LAN SYSTEM SW31RIO3
117         NICDEF 710 TYPE QDIO LAN SYSTEM SW32RIO3
118         MDISK 0100 3390 0001 12000 DISK012 MR ALL ALL ALL
119 USER QLS2RIO3 1G 1G
120         PROFILE MLNLINUX
121         NICDEF 700 TYPE QDIO LAN SYSTEM SW31RIO3
122         NICDEF 710 TYPE QDIO LAN SYSTEM SW32RIO3
123         MDISK 0100 3390 12001 24000 DISK012 MR ALL ALL ALL
124 USER QLS3RIO3 1G 1G
125         PROFILE MLNLINUX
126         NICDEF 700 TYPE QDIO LAN SYSTEM SW31RIO3
127         NICDEF 710 TYPE QDIO LAN SYSTEM SW32RIO3
128         MDISK 0100 3390 0001 12000 DISK013 MR ALL ALL ALL
129 USER QLS4RIO3 1G 1G
130         PROFILE MLNLINUX
131         NICDEF 700 TYPE QDIO LAN SYSTEM SW31RIO3
132         NICDEF 710 TYPE QDIO LAN SYSTEM SW32RIO3
133         MDISK 0100 3390 12001 24000 DISK013 MR ALL ALL ALL
```

Appendix I

Scenario III MLN project file

```
global {
  project scenario3
  $broadcast_address = 10.0.0.255
  $netmask = 255.255.255.0
}

superclass loadbalancer {
  template /dev/dasdb
  size 1G
  memory 512M
  network eth1 {
    netmask $netmask
    broadcast $broadcast_address
  }
}

superclass webserver {
  template /dev/dasdc
  size 20G
  memory 1G
  network eth0 {
    netmask $netmask
    broadcast $broadcast_address
    switch vsw11
  }
  network eth1 {
    netmask $netmask
    broadcast $broadcast_address
    switch vsw12
  }
  network eth2 {
    netmask $netmask
    broadcast $broadcast_address
    switch vsw13
  }
  network eth3 {
    netmask $netmask
    broadcast $broadcast_address
    switch vsw14
  }
  network eth4 {
    netmask $netmask
    broadcast $broadcast_address
  }
}

superclass sqlserver {
  template /dev/dasdd
  size 10G
  memory 1G
  network eth0 {
    netmask $netmask
    broadcast $broadcast_address
    switch vsw31
  }
  network eth1 {
    netmask $netmask
    broadcast $broadcast_address
    switch vsw32
  }
}

superclass loadbaltier1 {
  superclass loadbalancer
  network eth0 {
    netmask 255.255.255.0
    broadcast 192.168.1.255
    gateway 192.168.1.42
  }
}

superclass loadbaltier3 {
  superclass loadbalancer
  network eth0 {
    netmask $netmask
    broadcast $broadcast_address
  }
}

host LB1 {
  superclass loadbaltier1
  network eth0 {
    address 192.168.1.51
```

```
  }
  network eth1 {
    address 10.0.0.1
    switch vsw11
  }
}

host LB2 {
  superclass loadbaltier1
  network eth0 {
    address 192.168.1.52
  }
  network eth1 {
    address 10.0.0.2
    switch vsw12
  }
}

host LB3 {
  superclass loadbaltier1
  network eth0 {
    address 192.168.1.53
  }
  network eth1 {
    address 10.0.0.3
    switch vsw13
  }
}

host LB4 {
  superclass loadbaltier1
  network eth0 {
    address 192.168.1.54
  }
  network eth1 {
    address 10.0.0.4
    switch vsw14
  }
}

host LB5 {
  superclass loadbaltier3
  network eth0 {
    address 10.0.0.141
    switch vsw21
  }
  network eth1 {
    address 10.0.0.151
    switch vsw31
  }
}

host LB6 {
  superclass loadbaltier3
  network eth0 {
    address 10.0.0.142
    switch vsw22
  }
  network eth1 {
    address 10.0.0.152
    switch vsw32
  }
}

host WS1 {
  superclass webserver
  network eth0 {
    address 10.0.0.11
  }
  network eth1 {
    address 10.0.0.12
  }
  network eth2 {
    address 10.0.0.13
  }
  network eth3 {
    address 10.0.0.14
  }
  network eth4 {
    address 10.0.0.111
    switch vsw21
  }
}

host WS2 {
  superclass webserver
  network eth0 {
    address 10.0.0.15
  }
  network eth1 {
    address 10.0.0.16
  }
  network eth2 {
    address 10.0.0.17
  }
  network eth3 {
    address 10.0.0.18
  }
  network eth4 {
    address 10.0.0.112
    switch vsw21
  }
}

host WS3 {
  superclass webserver
  network eth0 {
    address 10.0.0.19
  }
  network eth1 {
    address 10.0.0.20
  }
  network eth2 {
    address 10.0.0.21
  }
  network eth3 {
    address 10.0.0.22
  }
  network eth4 {
    address 10.0.0.113
    switch vsw21
  }
}
```

```
host WS4 {
  superclass webserver
  network eth0 {
    address 10.0.0.23
  }
  network eth1 {
    address 10.0.0.24
  }
  network eth2 {
    address 10.0.0.25
  }
  network eth3 {
    address 10.0.0.26
  }
  network eth4 {
    address 10.0.0.114
    switch vsw21
  }
}

host WS5 {
  superclass webserver
  network eth0 {
    address 10.0.0.27
  }
  network eth1 {
    address 10.0.0.28
  }
  network eth2 {
    address 10.0.0.29
  }
  network eth3 {
    address 10.0.0.30
  }
  network eth4 {
    address 10.0.0.115
    switch vsw22
  }
}

host WS6 {
  superclass webserver
  network eth0 {
    address 10.0.0.31
  }
  network eth1 {
    address 10.0.0.32
  }
  network eth2 {
    address 10.0.0.33
  }
  network eth3 {
    address 10.0.0.34
  }
  network eth4 {
    address 10.0.0.116
    switch vsw22
  }
}

host WS7 {
  superclass webserver
  network eth0 {
    address 10.0.0.35
  }
  network eth1 {
    address 10.0.0.36
  }
  network eth2 {
    address 10.0.0.37
  }
  network eth3 {
    address 10.0.0.38
  }
  network eth4 {
    address 10.0.0.117
    switch vsw22
  }
}

host WS8 {
  superclass webserver
  network eth0 {
    address 10.0.0.39
  }
  network eth1 {
    address 10.0.0.40
  }
  network eth2 {
    address 10.0.0.41
  }
  network eth3 {
    address 10.0.0.42
  }
  network eth4 {
    address 10.0.0.118
    switch vsw22
  }
}

host SQLS1 {
  superclass sqlserver
  network eth0 {
    address 10.0.0.161
  }
  network eth1 {
    address 10.0.0.162
  }
}

host SQLS2 {
  superclass sqlserver
  network eth0 {
    address 10.0.0.163
  }
  network eth1 {
    address 10.0.0.164
  }
}

host SQLS3 {
  superclass sqlserver
  network eth0 {
    address 10.0.0.165
```

```
  }
  network eth1 {
    address 10.0.0.166
  }
}

host SQLS4 {
  superclass sqlserver
  network eth0 {
    address 10.0.0.167
  }
  network eth1 {
    address 10.0.0.168
  }
}

switch vsw11 { }
switch vsw12 { }
switch vsw13 { }
switch vsw14 { }
switch vsw21 { }
switch vsw22 { }
switch vsw31 { }
switch vsw32 { }
```