Virtualization, Isolation and Emulation in a Linux Environment

Jonas Eriksson

April 4, 2009
Master's Thesis in Computing Science, 30 hp
Supervisor at CS-UmU: Johan Tordsson
Examiner: Per Lindström

Umeå University
Department of Computing Science
SE-901 87 UMEÅ
SWEDEN

Abstract

Virtualization has received much attention the past few years. However, it is anything but new, as one of the first implementations were done in the 1960s to create a multi user system by running multiple instances of a single user operating system. Virtualization mechanisms have evolved much since that time, and an ever increasing number of similar technologies have been collected under the virtualization concept.

Several departments of the Swedish telecom company Ericsson are interested in exploring the use of virtualization for some of their products. In order to obtain a basic understanding of the technology, this thesis begin by exploring the history and theoretical foundation behind traditional virtualization. It then carries on by studying various virtualization technologies that do and do not fit into this definition. Next, it tries to answer the core question that both Ericsson and many other organizations ask themselves: Which virtualization technology is the best one? While the question can be answered for a certain application at a certain time, the general answer will for some time be that it depends on the problem at hand and what demands that exists for the technology, e.g. memory footprint, I/O performance, etc. The results from a series of generalized benchmarks are presented in order to enlighten the problems that exist with virtualization.

In addition to plain virtualization, common management tools for both the physical and virtual machines, with a natural weight on the virtual machines, are explored. By using the right type of management tool, the selection of virtualization technique need not be limited to one technique or technology.

Contents

Chapter 1

Introduction

The hardware used in computers today is still getting faster, and at the same time, the industry suffers from high demands to keep legacy software working and minimizing the amount of power consumed by computers. As virtualization allow for multiple Operating Systems (OSs) to run on a single piece of hardware at native or near-native speeds, many businesses introduces virtualization as a road towards solving these and other problems. A rich variety of virtualization software specially aimed at the GNU/Linux platform exist today, each technology targeting certain use cases. One of the current problems is that the virtualization concept includes a large variety of different technologies, and it is not always clear which technology to use in which case.

1.1 Problem Statement

There exist a vast number of virtualization technologies today. Some are aimed at solving specific problems and others claim to be of general purpose. Some of these technologies have been compared, but these studies are typically inconclusive regarding which technology truly is the best. Furthermore, existing studies lack metrics that are important to vendors like Ericsson. Selecting a specific technology for virtualization allow Ericsson to emphasize on product development and use less time to determine which technology to use when developing products. Ericsson has been using virtualization in different products in the past, but the choice of technology has most often been based on short micro-studies, and with a strong weight on what technologies developers are familiar with, since this has been what time has allowed.

This thesis takes a twofold approach, by doing both a thorough theoretical analysis and a series of tests and benchmarks of different virtualization and virtualization management solutions.

1.2 Purposes of the Thesis

By combining the original requirements with the remarks by the interviewees, described in detail in Section 3.1.1, a set of final requirements is constructed.

To ensure that this work is reproducible and to simplify studies in the future when new virtualization technologies become available, a tool is designed that allows for simplified benchmarking and testing. This tool is batch-oriented in order to require as little

human interaction as possible. In order to collect deeper information about the previous studies and of the requirements posed by Ericsson, interviews with developers and designers at different departments are performed.

The information contained in this thesis is to be *future proof*, in the sense that it describes technologies and applications in general, and focus less on certain virtualization products and technologies. This means that while the benchmark results and lists of virtualization technologies included are valid for the time being, the results should be seen as examples rather than eternal truths, as existing products develop quickly and new ones constantly emerge. The benchmarking results are to form the basis of a recommendation of the virtualization technology currently best suited for Ericsson's demands. If this is not possible, e.g. if one technology fulfill the demands of one area but fails at satisfying those of another, more than one technology may be recommended. The thesis should however strive to minimize the number of technologies recommended. As a final product for Ericsson, this thesis is summarized into an internal documentation of best practices for Linux virtualization. This contains not only the resulting recommendations of this thesis, but also some general guidelines, e.g. when to use virtualization, when not to use virtualization and the difference between different virtualization techniques.

As installations often are very large, it is important to investigate how to manage the virtualization technologies. Virtualization in effect adds a new layer between the OS and the hardware, which like the OS and the hardware needs to be managed in a scalable way. Because of this, management solutions are studied, by looking both at protocols and their implementations.

As a summary of the precise requirements, this thesis focuses mainly on capacity impacts, management flexibility, management completeness, and scalability of virtualization tools, but also the technical foundation, practical implementation, supported hardware platforms, configuration (the complexity thereof), community support, capacity impacts, and regulatory aspects. Current research in virtualization does not investigate the aspects of management flexibility and completeness.

Chapter 2

Virtualization

2.1 History and Formalizations

The exact origins of virtualization is unclear. One of the first documented use cases was computer researchers that wanted to move from batch processing machines towards machines able to simultaneously service multiple users with different demands. This was realized by e.g. the IBM VM/370 time-sharing system, whose evolution is thoroughly documented by Creasy [19].

2.1.1 The VM/370 Time-Sharing System

In the 1950s, computers were becoming more general in design, as opposed to their predecessors, that were most often designed for specific tasks. This generation of computers still ran large batch-type jobs that required the whole machine, and human interaction was needed for setting up new jobs, error handling, etc. This human interaction constituted of a relatively small part of the total execution time, since the computers at the time were slow. However, as computers became faster, the human interaction overhead became an even larger part of the time that the program needed to run.

An OS was introduced as an effort to lower the amount of human interaction that consumed valuable machine time by eliminating the interaction between programs. This however made developing programs tedious, as the computers became more isolated from its users. The turnaround time for a job submitted to a run queue could be hours or days. By the beginning of the 1960s, multi-user OSs were being researched. An OS developed at Massachusetts Institute of Technology, called the Compatible Time-Sharing System (CTSS) [19], was capable of running normal batch jobs, while at the same time allowing users to interactively access the same computer for execution of commands in real time.

In 1964, shortly after IBM announced its IBM System/360, IBM also announced a Control Program/Conversational Monitor System (CP/CMS) [19]. This enabled time-sharing by allowing more than one OS to run at the same time. This was much needed by the research community, allowing for more efficient development of among other things OSs. This system evolved into Virtual Machine Facility/370 (VM/370), which became available for the IBM System/370 in 1972. The VM/370 time-sharing system consisted of three parts. The Control Program (CP), which may be compared to the Virtual Machine Monitors (VMMs) of today, made one physical machine appear as one or more

copies of the same machine. These were referred to as Virtual Machines (VMs). The second component was the Conversational Monitor System (CMS), that was a single user OS capable of interactive use. Besides the CP and the CMS, a third component is included, the Remote Spooling and Communications Subsystem (RSCS). This is a communication system used for information transfer between OS instances [19].

The concept of the CP was introduced when trying to figure out what interface to present to programs running on the computers. By exposing the instruction set of the computer, each user essentially gets a private computer. In order to solve the problems of isolation between these VMs, two modes of execution were used: privileged and problem. In privileged mode, all instructions of the computer are allowed to be executed. In problem mode, on the contrary, only instructions that do not change or report the state of the machine are allowed to be run natively. These instructions instead transfer control to the CP, that reproduces the effect of the instruction, given that the VM has the sufficient rights to run the instruction [19].

2.1.2 Formalization of Requirements

The development towards platforms able to support instantiation into VMs continued, and was formalized in 1973, in a paper by Popek and Goldberg [58]. They define the state S of a computer as consisting of four components: executable storage E, processor mode M, program counter P, and a relocation-bounds register R that is used when accessing memory, in order to offset memory reads and writes.

$$S = \langle E, M, P, R \rangle$$

The notation $E[i]$ refer to the ith unit of storage in E, and $E = E'$ if and only if $E[i] = E'[i]$ for any i, $0 \leq i < q$, where q is the size of E. The relocation-bounds register $R = (l, b)$ consists of an offset l in E, which is a valid address and a size b of the memory addressable at the moment. If a memory access a is outside the size of E, i.e., $a + l \geq q$, or outside the memory addressable at the moment, i.e., $a \geq b$, the hardware issues a *memory trap*. A trap changes the mode of the processor to supervisor (see below) and switches control to the CP. Popek and Goldberg differentiate between *memory traps* and other *traps*, since a memory trap is more of an illegal instruction: To a program, the relocation-bounds register R should be viewed as the definition of the accessible memory. Attempts to access memory outside its defined bounds should be viewed as equivalent to a access outside the physical memory. By separating the memory traps, they do not disturb the rest of the definitions.

The mode M can be either supervisor mode (s) or user mode (u), corresponding to the privileged and problem-modes mentioned earlier. The program counter P is an address to a memory unit in E relative to the offset defined in R with the same restrictions applied as for memory access by the program itself.

An instruction i can be viewed as a function from one state to another. Given that the finite set of states is called C, i: $C \rightarrow C$. For example: $i(S_1) = S_2$. The instruction i is *privileged* if and only if for any pair of states $S_1 = \langle E_1, s, P_1, R_1 \rangle$ and $S_2 = \langle E_1, u, P_1, R_1 \rangle$ where neither $i(S_1)$ nor $i(S_2)$ causes memory traps, $i(S_2)$ traps and $i(S_1)$ does not trap. Note that the only difference between the states is the mode the processor is running in: For S_1, the processor is in supervisor mode, whereas the processor is in user mode in S_2. In other words, the instruction is privileged if the processor mode must be the supervisor mode in order to run the instruction.

The instruction i is *control sensitive* if there exist a state $S_1 = \langle E_1, M_1, P_1, R_1 \rangle$ and $i(S_1) = S_2 = \langle E_2, M_2, P_2, R_2 \rangle$ such that $i(S_1)$ does not memory trap, $R_1 \neq R_2$ and/or $M_1 \neq M_2$. That is, the instruction is control sensitive if it tries to change the amount of memory addressable and/or affects the mode of the processor.

The instruction i is *behavior sensitive* if the effect of its execution depends on the value of the relocation-bounds register (R) or on the mode (M) of the processor. The formal definition [58] is lengthy, and omitted for readability. An example of a behavior sensitive instruction is one that writes the current mode of the processor, M, to the memory. A *sensitive* instruction i is control sensitive and/or behavior sensitive. If an instruction is neither control sensitive nor behavior sensitive, it is *innocuous*. Popek and Goldberg [58] also outline three properties for a virtual machine:

> "The efficiency property. *All innocuous instructions are executed by the hardware directly, with no intervention at all on the part of the control program.*
>
> The resource property. *It must be impossible for that arbitrary program to affect the system resources, i.e. memory, available to it; the allocation of the control program is to be invoked upon any attempt.*
>
> The equivalence property. *Any program K executing with a control program resident, with two possible exceptions, perform in a manner indistinguishable from the case when the control program did not exist and K had whatever freedom to use privileged instructions that the programmer had intended.*"

The two possible exceptions mentioned are timing and resource availability. After all, as the computer is now shared between several programs, it is reasonable to look at the VM as a smaller version of the actual computer. As Popek and Goldberg [58] describe it: "logically the same, but with a lesser quantity of certain resources".

From the demands of the three properties and the formalization of the computer, Popek and Goldberg [58] formulates a theorem:

> Theorem 1. *For any conventional third generation computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.*

The term "third generation computer" refer to a machine that basically works according to the simplified formalization above. The theorem states that all instructions that change or read the mode of the processor or the amount of addressable memory must trap to the VMM. Popek and Goldberg [58] note that there exists very few computers that fulfill the requirements outlined above. Because of this, they outline the concept of a Hybrid Virtual Machine (HVM) by dividing the sensitive instructions into two (not necessarily disjoint) subsets. The instruction i is *user sensitive* if there exists a state $S = \langle E, u, P, R \rangle$ for which i is control sensitive or behavior sensitive. Conversely, with $M = s$, the instruction i is *supervisor sensitive* if there exists a state $S = \langle E, s, P, R \rangle$ for which i is control sensitive or behavior sensitive. This enables Popek and Goldberg [58] to formulate a theorem regarding hybrid VMMs:

> Theorem 3. *A hybrid virtual machine monitor may be constructed for any conventional third generation machine in which the set of user sensitive instructions are a subset of the set of privileged instructions.*

Popek and Goldberg [58] define the HVM as a VM where all instructions in virtual supervisor mode are interpreted, since the sensitive instructions cannot be trapped. As

an example, the PDP-10 instruction JRST 1 (return to user mode) is described. This instruction is supervisor control sensitive but not privileged. In other words, when executed from user mode, it yields no change or nor trap, as the processor already is in user mode. When running a VM in user mode, the VMM does not detect the transition from virtual supervisor mode to virtual user mode, making it unable to fulfill the equivalence property. This problem is solved by interpreting the instructions that are supposed to run in virtual supervisor mode.

2.2 Use cases

In essence, virtualization add a software layer between the hardware and the software that previously was used to communicate directly with hardware. This intermediate layer is leveraged by many applications of virtualization: features that are too complicated to be implemented in hardware may now be implemented in software while still being isolated from the OS. This section describes common use cases for virtualization.

2.2.1 Consolidation and Isolation

Server consolidation is an increasingly popular use case for virtualization, much because of the growing power saving demands on data centers. By running several underutilized server applications on one single physical server, the total amount of physical machines may be reduced. However, care must be taken when consolidating servers. The capability to handle bursts in load are limited, since the application is no longer alone on the physical server. Furthermore, the energy consumption of a server running several applications is higher than the same server running only one application. The savings in energy consumption may therefore not be as high as envisioned.

Live migration of VMs across physical machines, as described in Section 2.6.1, may be developed further by allowing a load balancing tool to transfer a VM when the load of the current physical machine gets to high. This could give consolidated servers the ability to deliver burst performance, given that the bursts of the VMs are interleaved in a way that allow them to be scheduled.

Isolation is a key part of consolidation and in high demand today, as this property enable hosting companies to let several users share one physical server. A popular way to create isolated machines today is through OS-level virtualization, as this technology provides isolation with a minimal overhead. As OS-level virtualization requires all isolated instances to run the same OS, it is not always the preferred technology.

2.2.2 Intrusion Detection

Smith and Nair [74] describe a Host-based Intrusion Detection System (HIDS) that run either as a part of a VMM, in a separate VM, or the same VM as the application being monitored. The HIDS can thus monitor the low-level activity of a VM or perform logging to a central intrusion detection system.

Having a monitoring application that is isolated or partly isolated from the application being monitored, increases the level of protection of the monitoring application. A root kit, i.e. a piece of program code that modifies the OS to hide itself and other applications (e.g. viruses), would be hard to detect using a monitoring application that runs on the same OS that it monitors. By moving the monitoring application outside of

the VM that is running the OS and giving the application monitoring abilities through the VMM, root kits would have a harder time hiding security breaches from the HIDS.

2.2.3 Debugging

As OS execution depends on external devices it is non-deterministic. Due to this, certain bugs are hard to reproduce. King et al [40] describe the Time-Traveling Virtual Machine (TTVM) as a VM where every state of the machine in a recorded time span is possible to examine. This is accomplished by saving the state of the whole machine at certain intervals (i.e. check pointing) and by recording all input sources. The recorded input sources makes the re-execution of the same code deterministic, as the re-execution no longer depend on external sources of information.

By adding TTVM support to User-Mode Linux (UML) and patching the debugging software gdb [32] to enable backwards stepping, King et al [40] create a OS debugger with support for deterministic execution replay. This is demonstrated by identifying four bugs, first by using the standard method that only allows for stack inspection and forward stepping. When this does not work or appears too complicated, the TTVM-capable gdb [40] is used to achieve a much faster and more efficient error isolation.

Whitaker et al [91] implements a TTVM-driven tool, Chronus, that can be used to debug erroneous configurations of systems. The check pointing is done in a more lightweight manner than described by King et al [40]: Chronus only record changes made to persistent storage. This provides lower overheads and makes the solution more suitable for production use. In order to find an erroneous change in configuration, Chronus replays states in a given interval. Next, testing for the erroneous state change is done by a probe, provided by the operator of Chronus. This probe is set to analyse the current state and indicate if it is faulty. The exact point of failure can then be located by Chronus using binary search.

2.2.4 High Availability and Fail Over

Tools that enable live patching of running kernels are becoming increasingly common [42]. By utilizing live migration, running kernels can also be migrated to different hardware with little or no downtime. This means that e.g. hardware upgrades and hardware replacement can be performed without service downtime.

Live migration on demand (e.g. for planned maintenance) is a well-researched topic. As expected, unscheduled hardware failure is a harder problem. To solve this, some vendors of virtualization technologies are promoting Storage Area Network (SAN)- or Network Attached Storage (NAS)-based solutions for VM-disks. Since SAN and NAS vendors often support replication of data, high availability services can be set up using nothing more than redundant copies of the hardware. Contemporary technologies can boot the virtual machine on the secondary hardware in the case of failure at the primary site. However, unlike live migration, the state of the VM is not preserved during this operation. It is therefore most suitable for stateless services, like web servers. A major problem for a stateful application like a database server is what happens when the network is segmented and e.g. the primary site continues to insert data into a database, while the secondary sites starts inserting data into the last known snapshot of the same database. As state-preservation sometimes is a requirement even when facing hardware failures, these mechanisms can fail to satisfy the availability needs of some applications. Since a state-preserving fail over techniques requires the content of the whole RAM to

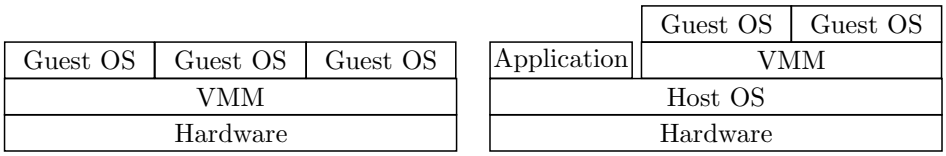

Figure 2.1: Type I VMM (left) and type II VMM (right).

exist in two places at the same time, this ability is not realized with the use of VMMs alone.

2.3 Techniques

2.3.1 Full Virtualization

The properties listed in Section 2.1.2 define the modern view of a VMM or *hypervisor* capable of full virtualization; the goal is to run as many instructions as possible on the bare hardware in order to maximize the performance of a VM.

Full virtualization includes both VMMs that operates according to Theorem 1 and those that adhere to Theorem 3 in Section 2.1.2. The most likely reason for not differentiating between these two types of VMMs is the lack of hardware platforms that adheres to Theorem 1. For example, as the analysis by Robin and Irvine [70] conclude, the popular Intel Pentium (x86-line) lack the ability to sustain a secure VM. As the x86-line of processors is one of the most popular to use in servers, one of the most common usages of virtualization today, the difference between the two classes is ignored. However, the two major manufacturers of x86 processors, Intel and AMD, have both announced hardware support for virtualization [5, 38]. Their respective approaches are further explored in Section 2.4.

One does however differentiate between *type I* and *type II* VMMs. The difference between the two is in management of the hardware. As illustrated in Figure 2.1, the type I virtual machine VMM runs directly on hardware, like the CP described in Section 2.1.1. In the type II virtual machine, also illustrated in Figure 2.1, the VMM run on top of an host OS, that handles resource allocation and communication with hardware.

2.3.2 Para-virtualization

As most hardware platforms adhere to Theorem 3 by Popek and Goldberg [58], the virtual supervisor mode have to be emulated. Even if the hardware platforms adhere to Theorem 1, trapped instructions has to be emulated by the VMM. This emulation would most often incur a large overhead: The instruction must be fetched and decoded, the (potentially large) result may have to be moved to the VM's memory location, etc. As demonstrated, e.g. by King et al [39], this overhead may be mitigated by utilizing *para-virtualization*. The basic idea is to make the OS running in the VM (henceforth called the *guest OS*) aware of the fact that it is running in a VM. As para-virtualization breaks the equivalence property defined by Popek and Goldberg [58], it may not always be an acceptable solution. However, the ability to speed up e.g. the execution of I/O operations may be vital in some virtualization use cases.

As described by Amsden et al [6], para-virtualization uses the notion of a *hyper call*. The hyper call is inspired by system calls in OSs. A system call is invoked by

a program running in non-privileged mode on the processor when it want access to a resource managed by the OS, i.e. a privileged resource. Unlike a system call, that is made to the OS, the hyper call is made to the VMM. The VMM handles the request much like an OS handles a system call. This enables the guest OS to e.g. schedule and queue requests, and benefit from the fact that it is running in a VM to make better decisions regarding scheduling, etc. King et al [39], among others, show that this yields large performance improvements in some cases. As an effort to simplify implementations of para-virtualization, Amsden et al [6] propose a standardized hyper call interface.

2.3.3 Hybrid Para- and Full Virtualization

Popek and Goldberg [58] define hybrid virtualization as a virtualization technology that emulates instructions that run in virtual supervisor mode, as summarized in the end of Section 2.1.2. However, this difference is not noted in contemporary virtualization contexts. Nakajima and Mallick [55] redefine hybrid-virtualization as a mix between para-virtualization and full virtualization using hardware support, the latter outlined in Section 2.4. In pure para-virtualization, all calls in the OS that would or should trap to the VMM has to be performed using hyper calls. Although the use of para-virtualization can yield higher performance, e.g. for I/O operations, it does not make hyper calls the most efficient solution to replace all sensitive instructions. By utilizing hardware support for virtualization, Nakajima and Mallick [55] both reduce the changes required in the code base of the guest operating system, and achieved a more efficient virtual machine than would be possible through pure para-virtualization.

2.3.4 OS-level Virtualization

One use case for virtualization is resource isolation between users. In this case it might be appropriate to, instead of running a number of VMs that in turn run multiple copies the same OS kernel, create isolated process contexts inside one OS kernel. This technique is formalized and implemented for NetBSD by Džonsons [28]. Implementations of OS-level virtualization for Linux exists, and are described in Section 2.6.

OS-level virtualization has became increasingly popular lately: A large number of hosting providers are offering Virtual Private Servers (VPSs), which is the name most often used to promote products based on OS-level virtualization. The popularity of this technique is most likely due to the low overheads gained from not requiring a VMM layer between the hardware and the user kernel. Since OS-level virtualization lack the VMM found in e.g. full virtualization, the isolated instance correspond to a VM is referred to as a *container*.

2.3.5 Hardware Emulation

With virtualization, the focus is to provide VMs that are replicas of the physical machine on which the VMM run. The requirements might however be different in e.g. a development environment, or when legacy machines still in production environments fail without possibility of repair or replacement, i.e., the platform that the program is built for might not be physically available. Because of these requirements, hardware emulators are most often aimed at virtualizing not only parts of the hardware, but whole machines. A successful example of this is the Multiple Arcade Machine Emulator (MAME) [73], that emulates legacy arcade machines.

As Smith and Nair [74] describe, hardware emulation is most often implemented using either interpretation or binary translation. The former is mostly popular in development environments where the operation needs to be performed exactly as it would have been on the real physical hardware, i.e. simulation is required. By reading and interpreting the instruction, the effect of the instruction can be performed on a structure that contains the VM. As a consequence of the correctness of operation in these environments, the developer is able to observe the state of the VM when executing a instruction. The most popular method, used when only the result of a program, as well as speed and low overhead is of interest, is binary translation. By interpreting the instructions about to be executed, the emulator may convert the instructions to the instruction set of the target machine. Often, this is done by a caching adaptive binary translator, that reads instructions until either a branch is encountered or a certain number of instructions have been read. These are then translated passing a dynamic optimizer, making the resulting code as efficient as possible. By also using a cache, already translated code blocks may be reused if they reappear.

2.4 Hardware Support

As concluded by Robin and Irvine [70], the Intel Pentium is unable to sustain a VM in a secure manner while conforming to the requirements stipulated by Popek and Goldberg [58]. These requirements are reformulated by Robin and Irvine [70] to fit the type I and type II VMM distinction.

Recently, both Intel and AMD have developed virtualization support in their line of 64-bit x86 processors. These technologies are called VT-x and AMD-V and are documented in work by Neiger et al [56] and Zeichick [95, 96], respectively. These technologies are basically similar, and operates by adding another dimension to the privilege levels of the processor. In similarity to the machine formalized by Popek and Goldberg [58], the x86 processor has multiple privilege levels, commonly referred to as *rings*. There are four rings, with ring 0 having the highest privileges and ring 3 having the lowest ones. Traditionally, OSs run in ring 0 and applications run in ring 3. When using a VMM, however, the VMM must run in ring 0. As noted by Robin and Irvine [70], a OS running inside a VM in ring 1, 2 or 3 does not fulfill the requirements documented by Popek and Goldberg [58].

By adding this new dimension to the privilege levels, processors are able to switch not only between rings but also between a VMM mode and a less privileged VM mode. This allow the guest OS to run in ring 0 with its applications in ring 3, but inside the less privileged VM mode. System calls from the applications cause a trap directly into the guest OS instead of taking a detour via the VMM as it would have to do without the virtualization extension.

The development of these technologies is currently moving quickly forward. Features implemented by one technology is most often scheduled for inclusion in the other and vice-versa. Some of the features expected by each technology are para-virtualization support via a hyper call instruction and an I/O Memory Management Unit (IOMMU) that makes it safer to dedicate Direct Memory Access (DMA) capable hardware to a VM.

2.5 Research

The current virtualization technologies are typically based on earlier implementations with roots in the research community, exemplified by the wide range of research based on the Xen project [18, 31, 52, 54, 55, 85, 97].

Hand et al [35] explore the connection between VMMs and microkernels, a popular research model of OSs. Microkernels are built around the idea of the OS being communication oriented with a modular design, and uses message passing interfaces to communicate between the different modules of the kernel. The core kernel should work only like a hub in the message passing system. The claimed advantage is that the minimalistic and modular design of the core kernel and the surrounding components makes the kernel easier to maintain and port to different architectures.

Hand et al [35] point out that the minimalistic design of the VMM, probably referring to type I VMMs, has a lot of similarities with the design of microkernels, and even add some improvements. For example, a microkernel OS depends on user-mode applications to carry out central services in the kernel, e.g. paging. Xen solve this by having the VMM ignore paging completely, and assign each VM a partition of the memory and thus make the guest OS responsible for paging.

Since VMs running on a VMM most often are self-contained, the need for Inter-Process Communication (IPC) between these are not as vital as in a microkernel OS. Some VMMs, e.g. Xen, support high bandwidth communication by device-channels. These enable high bandwidth communication between VMs, while still maintaining the isolation required in both VMMs and microkernels.

A problem often faced by new microkernel OSs, as well as other new OSs, is that the OS often introduces a new Application Programming Interface (API) to application software. This often force developers of microkernel OSs to write emulation layers to be able to support applications developed for other OSs. With VMMs, however, the ability to run an unmodified application on top of a guest OS is provided. This is often one of the most compelling use cases of VMMs. By noting these and some other similarities and differences between microkernels and VMMs, Hand et al [35] conclude that VMMs can be thought of as “microkernels done right”.

2.5.1 Real-Time Scheduling and Embedded Systems

Similar to other systems, embedded systems are becoming faster and getting more cores. The requirements are however somewhat different from those of ordinary servers. IPC between systems is often higher in demand, since embedded systems often consist of several systems that co-operate. Some of these systems may also require real-time scheduling, which may be hard to accomplish through the abstraction that a VMM create.

Augier [7] raise the point of OS capabilities: While Real-Time OSs (RTOSs) are aimed at small footprints and real-time scheduling, requirements from increasingly popular rich environments such as mail clients and web browsers must be considered. While one approach is to extend the RTOS to support such environments, another is to adapt an existing General Purpose OS (GPOS) already capable of providing a rich environment, like Linux, and extend this GPOS to fulfill the demands of real-time scheduling. Extending this second approach, one could let one or more RTOSs co-exist with one or more GPOSs, to benefit from the properties of both OS types. This technique may also be used to simplify the future use of legacy applications and OSs.

Providing real-time scheduling through the abstraction of a VMM may be hard to accomplish. Augier [7] propose a Scheduler Virtual Device (SVD), that handle all scheduling decisions for guest OSs running on the VMM, regardless of the OS type. In order to achieve this, OSs running on the VMM are para-virtualized and update the SVD continuously with information such as creation, blocking, unblocking, and destruction of threads through a driver interface called a Scheduler Interface (SI). This allow the SVD to have a global view of the scheduling needs of the guest OSs.

2.6 Contemporary Virtualization Technologies

As a complement to the summary of virtualization techniques, a number of technologies and their respective features are described below. A summary of these technologies is found in Table 2.1. The amount of information available differs a lot between technologies, and because of this, the amount of information in the descriptions varies.

2.6.1 Xen

Xen is a VMM originally developed at the University of Cambridge and is outlined by Barham at al [8]. Thanks to its thorough documentation in manuals and scientific papers as well as its open source code base, Xen is a popular target for scientific research. Many of the ideas implemented using Xen have made it to the release version of Xen. The Xen VMM expose a para-virtualization interface towards the guest OS, and therefore require the guest OSs to be ported to the platform. One VM instance, referred to as domain zero or dom0. From this VM, all aspects of the VMM may be controlled. Less privileged VMs is referred to as user domains or domUs.

As virtualization introduce an abstraction of hardware, it was realized that this could be used to decouple OSs from their hardware, making it possible to migrate VMs live over a network connection to another physical host. Clark et al [18] describe the theory behind the Xen implementation of live migration, which is based on adaptive algorithms that transfer memory pages made dirty since the previous transfer. By increasing the amount of bandwidth used in each transfer, the transfer of the least often used pages does not interfere with normal operations, and only pages that are dirtied often have to be transferred using the maximum bandwidth.

In an effort to combine the speed of communicating directly with hardware devices and the isolation of hardware abstractions, Fraser et al [31] describe an implementation of a safe device driver model that isolate the driver in an I/O space. The guest OS then communicate with the driver using a device channel. The isolation of this model allows the driver to crash without crashing the guest OS. This is demonstrated by running a disk driver in an I/O space while both reading and writing the disk. While the device restart does not crash the guest OS, it incur a small delay in device access.

The live migration described by Clark et al [18] and the hardware abstraction introduced by Fraser et al [31] are two examples of features included in the 3.0 release of the Xen project (this release includes the ported Linux kernel). In addition to these, support for the AMD64/EM64T platform is described in detail by Nakajima et al [54], and hybrid para- and full-virtualization is covered in length by Nakajima and Mallick [55]. A broad summary of recent Xen development is given by Pratt et al [59]. Even after the 3.0 release of Xen the contributions have been plentiful, including better Symmetric Multiprocessing (SMP) performance by Theurer et al [85], improved network performance by Menon et al [52], a method utilizing functions already present in the

Linux kernel for doing live migration while using a dedicated non-abstracted hardware device by Zhai et al [97], to mention a few.

Several virtualization technologies are making it into the official releases of the Linux kernel, and Xen is no exception. As of version 2.6.23, support for running Linux kernels as user domains is included using the kernel facility paravirt_ops [17]. Support for dom0 capabilities are currently developed by Xen maintainers, as the dom0 capabilities is a superset of the domU capabilities. The dom0 support is scheduled to be included in Linux kernel version 2.6.29 [17].

2.6.2 KVM

The Kernel-based Virtual Machine (KVM) is an effort to create a VM environment handled by the Linux kernel. KVM is included in the official stable Linux kernel releases since version 2.6.20 (released Feb 4 2007 [46]), but development patches exists for the 2.6.16 kernel and forward [63]. Kivity et al [41] describe KVM as a Linux subsystem that adds VMM capabilities to the Linux kernel. This is realized by adding support for the Intel VT-x and AMD AMD-V virtualization hardware. By exposing a device file, `/dev/kvm`, user-space programs may use the kernel facilities to create and modify a VM. KVM actually performs as little work as possible, and lets the user-space program that created the VM handle e.g. I/O and signals. This is done using a modified version of QEMU (see Section 2.6.6), that is capable of using the `/dev/kvm` device to improve performance of the VM when possible.

Qumranet inc., that have been developing KVM, was recently acquired by Red Hat inc. [67]. The large developer organization of Red Hat, combined with the inclusion in the upstream Linux kernel, have made KVM a fast growing project. KVM currently features live migration and support for both SMP hosts and guests, as well as para-virtualized block- and network devices [63, 64, 65].

2.6.3 Lguest

Lguest is a simple x86-only para-virtualizing VMM that is aimed towards performance and has a smaller and less complex code base than e.g. KVM. As Lguest is using para-virtualization, it can only run Lguest-compatible Linux kernels. It does however not require the processor extensions that are needed by e.g. KVM. At the time of writing, Lguest only supports 32-bit kernels. There is a 64-bit capable version, although this is experimental. Lguest is included in the official stable Linux kernel since version 2.6.23 [71, 72].

2.6.4 UMLinux

As with Xen, the development of UMLinux started in the academia. The motivating scenario was to run experiments with fault injection. UMLinux is designed as a type II para-virtualized VM, with a modified Linux kernel running as a user space process. The hardware is abstracted using device drivers similar but not identical to the underlying hardware [39]. King et al [39] describe improvements of some aspects of UMLinux such as context switching, making its performance comparable with that of other virtualization technologies.

2.6.5 UML

User-Mode Linux (UML), not to be confused with UMLinux or the Unified Modeling Language, is described by Dike [22] as a Linux kernel ported to be run on a Linux kernel, effectively creating a para-virtualized user-mode VM. The main difference between UML and UMLinux as noted by King et al [39], is that UML uses a separate host process for every process in the guest OS whereas UMLinux isolates all guest OS processes inside the process running the guest OS. The benefit of the former approach is that it is simpler for guest OS processes to access resources in the host OS, whereas fair scheduling and isolation between VMs is harder. UML is included in the Linux kernel since the release of 2.6.0 [66].

2.6.6 QEMU and KQEMU

QEMU is a multi-platform emulator with the ability to run both user-mode applications and OSs on x86, AMD64/EM64T, and PPC hardware, while emulating a number of platforms, among others x86, AMD64/EM64T, PPC, SPARC, MIPS and m68k. Support for more host as well as guest platforms are currently being developed [10]. QEMU utilize dynamic binary translation [9], covered briefly in Section 2.3.5. The project include a host driver, KQEMU, that provide near native performance [9]. This driver make QEMU a type II full virtualization technology.

2.6.7 Wind River Hypervisor

Wind River is a long time manufacturer of embedded solutions, e.g. the embedded RTOSs VxWorks and Wind River Linux. The Wind River Hypervisor is designed for embedded systems, to work well with mainly these OSs, and has support for running RTOSs on top of the VMM. In addition to having a low memory footprint, the Wind River Hypervisor focus on providing different solutions for IPC between VMs with different target applications [93].

2.6.8 OpenVZ

OpenVZ is an OS-level virtualization technology distributed in the form of a series of patches to the Linux kernel. Its development is supported by Parallels as part of the company's OS-level virtualization product Virtuozzo. OpenVZ is claimed to be highly scalable with strong isolation between containers [83]. The OpenVZ team is currently contributing to the upstream Linux kernel with patches for resource management using *bean counters* [29]. The bean counter model is central in OpenVZ to manage resource allocation to different VMs, and works by limiting the number of resources one process or a group of processes may use. These resources may be memory, shares of the processor, etc.

2.6.9 LXC

Linux Containers (LXC) [20] is an effort aimed at creating a OS-level virtualization technology fitted to be included in the mainline Linux kernel. The LXC patches are today included in the *-mm* series of patches to the Linux kernel, which is a collection of experimental patches [47]. The functionality is still experimental, but having a single OS-level virtualization technology for Linux should help unify the efforts of the community.

2.6.10 Linux-VServer

Linux-VServer is another OS-level virtualization project aimed at taking advantage of already existing security systems in the Linux kernel [48]. Camargos et al [43] note that Linux-VServer scales poorly when running the SysBench database benchmark on more than four VMs. This problem show that general benchmarking may be deceiving, as Linux-VServer performs well in similar tests.

2.6.11 Solaris Containers

As of Solaris 10, support is included for OS-level virtualization with Solaris Containers or *Zones*. This feature is enabled by default when booting a Solaris installation, with all processes running in the *global zone*. When a new zone is created, new processes may be started in this *non-global zone* instead of in the global zone. The original design assume that the directories `/usr`, `/lib`, `/platform`, and `/sbin` are shared between all zones, but in practice each container may have its own file system [80].

2.6.12 Sun xVM Server and Sun xVM VirtualBox

Sun xVM is a virtualization suite that contains an administration tool (xVM Ops Center) [78], a type I VMM aimed at server use (xVM Server) [77], a type II VMM aimed at desktop use (xVM VirtualBox) [76], and software to remotely access desktops running on VMs [79]. Both xVM Server and xVM VirtualBox support para-virtualized drivers inside the guest operating system in order to access hardware more efficiently.

2.6.13 VMware ESX, VMware ESXi, and VMware Workstation

VMware ESX and VMware ESXi are type I VMMs aimed at server environments capable of e.g. live migration using VMmotion and booting VMs from network attached devices [86]. The difference between the two is that VMware ESXi is a more lightweight implementation, lacking the *service console* included in VMware ESX [88]. VMware Workstation is a type II VMM aimed at desktop use [89]. All three technologies support para-virtualized drivers inside of the guest OS.

2.7 Benchmarking

Research concerned with improving existing virtualization software are often focused on a small part of the VMM, and the focus of the testing is thus often narrowed down to the modified parts and possibly also the parts that may be affected by the modification. When improving and therefore also testing the scalability, the usual setup is to perform a given task in parallel on a number of VMs, and comparing the result with the same task running in parallel without the overhead of a VMM [85]. When improving execution speed, only one VM is used and a comparison is done to a setup with an identical machine without a VMM [31, 4, 55]. This method is used also when comparing performance among VMMs. The results are either compared only to other VMMs [39, 43] or to the performance obtained using only the guest OS without virtualization [8]. Comparative tests of scalability most often use the same method as when focusing on scalability improvements. In this case, the speed is often normalized with the same amount of work being run on the guest OS without virtualization in order to compare different technologies [21].

Table 2.1: Summary of contemporary virtualization technologies.

Implementation	Technology
KQEMU [9]	Type II full virtualization
KVM [63]	Type II full virtualization
Lguest [72]	Type II para-virtualization
Linux-VServer [48]	OS-level virtualization
LXC [20]	OS-level virtualization
OpenVZ [83]	OS-level virtualization
QEMU [9]	Hardware emulation
Solaris Containers [80]	OS-level virtualization
Sun xVM Server [77]	Type I full virtualization with para-virtualization support
Sun xVM VirtualBox [76]	Type II full virtualization with para-virtualization support
UML [22]	Type II para-virtualization
UMLinux [39]	Type II para-virtualization
VMware ESX & ESXi [86, 88]	Type I full virtualization with para-virtualization support
VMware Workstation [89]	Type II full virtualization with para-virtualization support
Wind River Hypervisor [93]	Type II full virtualization with para-virtualization support
Xen [94]	Type I full- or para-virtualization

An example of a highly specialized test of a very specific feature is described by Clark et al [18], that measure the efficiency of a live migration feature implemented for the Xen VMM. As a basis of the implementation, a tracking of the working set (by measuring the number of pages made dirty per second) was first performed in order to figure out if the pages may be transferred dynamically. After implementing the solution, measurements of bandwidth usage was done to improve the adaptive page-transferring algorithm. The implementation was then tested both by running traditional tests as SPEC CINT2000, a Linux kernel compile, the OSDB OLTP benchmark with PostgreSQL, and SPECweb99 using Apache, but also by running a Quake 3 network server, all while migrating. The degradation of service in all cases were measured to be rather low, with players connected to the Quake 3 server being totally oblivious to the fact that the VM running the server was migrated to a different physical machine [18].

2.7.1 SPECvirt_sc2009

An effort to create a standardized metric for comparing VMMs, called SPECvirt_sc2009 is currently being performed by the SPEC Virtualization Committee [75]. Theurer et al [84] describe the current status of this benchmark, but point out that the information is subject to change since it is still being developed. The benchmark is based on the *tiles* concept described by Makhija et al [49] as a part of the VMmark benchmark. The tiles are similar to the workload concept described by Casazza et al [13] as a part of the vConsolidate benchmark.

The concept of tiles is aimed at measuring the VMM's ability to consolidate servers. The metric used is how many tiles that can be run simultaneously while maintaining

Table 2.2: Roles and applications used in one SPECvirt_sc2009 tile.

Guest	Application
Web server	SPECweb2005
Mail server	SPECmail2009
Application server	SPECjAppServer2004
Database server	Serves the application server using Besim
Infrastructure server	Serves partly the web server using NFS
Idle server	–

a certain quality of service from the services running in the VMs. One tile consists of six guests: a web server, a mail server, an application server, a database server, an infrastructure server, and an idle server. All these, except the idle server that represents the overhead of just running a VM, run SPEC standard tools for benchmarking systems detailed in Table 2.2. In order to make one tile represent a static unit of work, the standard benchmark tools are modified to only do a certain amount of work.

Unlike most other benchmarks, SPECvirt_sc2009 test VM to VM network communication. By making the application server rely heavily on the communication to the database servers, and the web server rely partly on the infrastructure server, VM to VM communication constitutes a significant part of the benchmark.

2.7.2 Time

When benchmarking the performance of VMs, it can be hard to use tools traditionally used for benchmarking. As traditional benchmarking tools typically perform an action and measure the time the action takes, or conversely measure how many times an action can be performed in a certain amount of time, the demand for precise time keeping is key. However, in a fully virtualized environment, running time-keeping interrupts on the virtual processor is determined by scheduling factors in the underlying hardware. When using para-virtualization, the time is usually synchronized using a hyper call which allows for more precise time-keeping. Despite this, some problems with time drift have occurred in the past, as documented [69].

In order to avoid problems related to timing difficulties, benchmarks may either focus on quality of service aspects like SPECvirt_sc2009 does, or run tests with long duration, e.g. building the Linux kernel. In these experiments, the system clock is synchronized before and during the test using e.g. the Network Time Protocol (NTP) [37]. Tests that require high precision time are however unsuitable for fully virtualized environments.

Another possible solution to the timing problem is to use a non-virtualized client that perform the time keeping. When e.g. testing network bandwidth, the values obtained from such a client should be more reliable than values measured inside the VM.

2.8 Management

The problems of vendor lock-in and system heterogeneity have existed for a while, and as virtualization is becoming more common in production environments, organizations not only have to think about how to handle different hardware, OSs and applications, but also different VMMs. Efforts have been made to unify not only the management of VMs, but also that of VMMs and physical servers.

2.8.1 CIM and VMAN

The Common Information Model (CIM) [23] is an open standard that defines a number of classes and objects that can be used to model management information about systems, networks, applications, and services. By storing management information using a common model, converting between different formats implementing that model is much simpler and vendor lock-in is avoided. It is important to note that CIM is an abstract format described using the Unified Modeling Language (UML), and only define how information should be stored conceptually. It is also important to note that UML is defined using the Meta-Object Facility (MOF) specification developed by Object Management Group (OMG), whereas CIM classes are written in the Managed Object Format (MOF) syntax language defined within CIM, which are both developed by the Distributed Management Task Force (DMTF). Although Meta-Object Facility and Managed Object Format share acronyms, they are not related.

As CIM only abstractly define the managed objects, a way to communicate with, and about these objects (or with a CIM Object Manager (CIMOM)) has to be created. DMTF suggest an implementation of CIM called Web-Based Enterprise Management (WBEM) [26]. This implementation uses a standard called CIM-XML [25], that is used to represent CIM classes using Extensible Markup Language (XML), and a Simple Object Access Protocol (SOAP)-based protocol called Web Services for Management (WS-Management) [27]. As WBEM is an unifying management system, The Open Group has proposed a standardized C API called Common Management Programming Interface (CMPI) [82], which create a CIM provider, through which non-CIM systems may be managed using WBEM management tools.

As virtualization is becoming more and more common, and as CIM was created with physical servers in mind, DMTF has started the Virtualization Management (VMAN) initiative [24]. This define not only how to store management and resource information about virtualized systems, but also define a format for storing images of VMs called the Open Virtualization Format (OVF). The relationships of most of these standards are further described by the diagram in Figure 2.2.

OVF is less abstract than the CIM parts of VMAN, as it defines how the data should be stored on disk, using a file structure and XML, that may be packaged in a Tape Archive (TAR). It provides details on what resources the VM has, what the included disk image(s) contain, and other information that enables the migration of a VM contained in a OVF file. OVF does however not define the format of the virtual disks, but does demand that an URI to a document describing the disk format is specified.

2.8.2 SNMP and NETCONF

The Simple Network Management Protocol (SNMP) first appeared as a series of Request For Comments (RFCs) in 1988 [51, 50, 15], and was intended to be used as a protocol to monitor and configure hosts. The version published in 1988, commonly referred to as SNMPv1, did not provide any facilities for authentication; the only means of providing this consists of a *community*-string that is sent along with the request. A solution to this and other extensions are introduced in later versions, and SNMPv3 [36, 14, 45, 12, 92, 62, 61, 60] provides both authentication, privacy and access control. Since SNMPv3 is not backwards compatible in the traditional sense, best practices to help vendors to allow multiple SNMP versions to co-exist have been documented [33].

Despite the wide range of usage envisioned, the usage of SNMP today is mostly limited to monitoring. As an effort to create a more specialized networked management

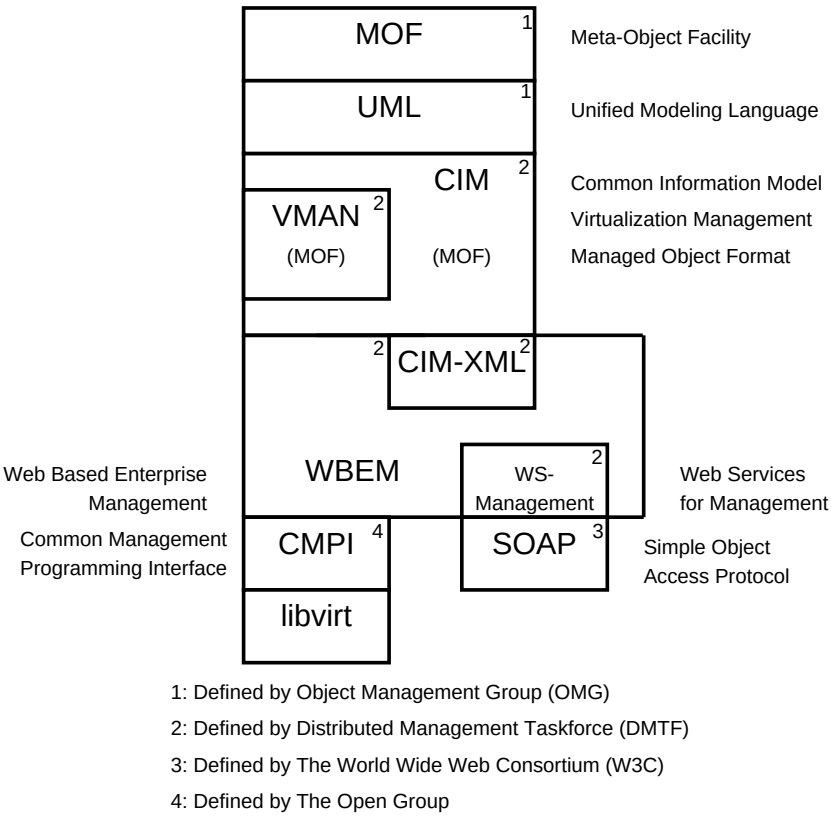

Figure 2.2: Conceptual model of management technologies and related protocols.

solution, the Network Configuration Protocol (NETCONF) [30] has been developed. Unlike SNMP, NETCONF only defines the interchange of commands and information encoded in XML. The transport of said information, as well as authentication, privacy and access control, is handled by existing protocols such as Secure Shell (SSH) [90], SOAP [34] (with Secure Socket Layer (SSL)), and Blocks Extensible Exchange Protocol (BEEP) [44]. Like SNMP, NETCONF may be partitioned and simplified, and the server embedded in the managed device can be made quite simple if necessary. An effort is currently ongoing to create a data modeling language for NETCONF schemata, called YANG [11], corresponding to Management Information Base (MIB) in SNMP and Managed Object Format in CIM.

Despite the popularity of SNMP and that the problems with SNMP are solved in NETCONF, neither of these protocols are popular for management of VMMs. Whereas larger vendors, e.g. VMware [87] supply optional SNMP support, many other efforts, e.g. SNMP-support for Xen [16], are community-driven. To the best of our knowledge, no vendor supports NETCONF.

2.8.3 libvirt

The libvirt [1] specification is an effort to create a common management API for VMMs started by Daniel Veillard at Red Hat in the end of 2005. The project started out as

libxen, but quickly changed name to libvir and became known as libvirt in February 2006 [53]. The implementation libvirtd, also developed by the libvirt community, is shipped with most Linux distributions that support virtualization e.g. SuSE Linux Enterprise Server [57] and Red Hat Enterprise Linux [68].

In addition to management and monitoring through a variety of drivers for different VMMs, libvirt is capable of storing persistent information about VMs and virtual networks using a VMM-independent XML-format that is mapped to the configuration format used in the VMM by libvirt [2]. In order for libvirt-managed VMMs to be available to a CIMOM, a CIM provider called libvirt-cim [3], that conform to the API defined by CMPI [82], has been developed.

Chapter 3

Results

3.1 Preliminaries

3.1.1 Investigation of Existing Attitudes Towards Virtualization

During the theoretical in-depth study and shortly thereafter, several interviews with developers and management staff at Ericsson have been conducted. As this thesis was done at a section of Ericsson that develops and maintains platforms for use in the products Ericsson sell, input from the internal customers is an important part of the thesis. As it turns out, virtualization is already in use in products and testing environments. Plans exist for the use of it in future products motivated by many of the traditional reasons for virtualization, e.g. consolidation and isolation. Following is a summary of the interviews. As Ericsson products where discussed during these interviews, a complete transcript cannot be included.

Questions Posed by Interviewees

In order to collect information about the preferred scope of the thesis, interviewees were asked what question(s) they wanted the thesis to answer. The number one question expressed is "Is virtualization the right way to solve the problem at hand?". This is tightly coupled with the concern that virtualization is used for the wrong reasons. Two other concerns were about VMM taxonomy; "How similar are different technologies?", and regarding security: "How secure is a VMM?".

Usage Cases

Consolidation is as noted a common use case for virtualization. However, in the telecom industry, fault tolerance demands are often high. As an example, consider System A, that requires five physical servers to fulfill a certain fault-tolerance requirement. This is a very common demand, as requirements may stipulate that e.g. failure of one hardware unit may only result in the loss of one fifth the capacity of a system in order for it to conform to the requirements. The need for five physical servers creates a threshold that makes it harder for small customers to invest in technology that fulfill this type of fault-tolerance requirement.

In order to support both large and small customers with the same system, System B1 may be created. System B1 uses the same software as System A, but the five physical

servers are replaced with one physical server running five VMs. This means that System B1 requires one fifth of the hardware System A require, and System B1 may be sold to smaller customer while still using the same software as System A. However, this means that System B1 no longer fulfills the fault-tolerance requirement that the original system was designed for.

To create a system that fulfill the same fault-tolerance requirement as System A, five physical servers are obviously required. By allowing the hardware to be shared, a customer may buy the same hardware as System A uses, and use this hardware to host five instances of System B2, that run on VMs like in System B1, but with all VMs of one instance of System B2 running on separate physical machines. The owner of the hardware can then re-sell five instances of System B2 to smaller customers. This way, System B2 can be offered with a smaller total capacity (and thus cheaper), while still fulfilling the fault tolerance requirements of the larger and more expensive System A.

Isolation and consolidation may also be used in testing environments. As deployed networks may consist of a rather large number of nodes and links, the ability to simulate such a network instead of buying the hardware can result in large savings. The virtualization of hardware and networks may even make it easier to find the source of errors, as deep inspection of hardware states and link traffic is simplified.

Since development and debugging are closely related, virtualization may be used in the same way when developing as when debugging. This usage can also bring the two areas closer together, as it is easier for a developer to have a personal system where certain features may be tested immediately after implementation. For debugging, solutions like the TTVM described in Section 2.2.3 may be of special interest.

When correcting errors in software, an important part is to avoid introducing new problems by modifying the environment in a way that affects the behaviour of the software. For example, if a binary is compiled from some source code, the same source code should generate an identical binary when compiled again. This can be achieved by running the build environment as a virtual machine that may be check pointed before a certain build, making the exact build environment used readily available if some bug in a certain build has to be reproduced.

Maintaining legacy systems in an otherwise evolving environment may be one use of virtualization. The opinions in this matter is however divided. Most interviewees believe that a better solution is to replace the legacy system. Whereas replacement is a viable long-term solution for avoiding problems with legacy systems, it is typically costly. Possibilities to develop a new system may be hindered by time and budget constraints.

Problems

There is some skepticism towards para-virtualization, with the most common concern being that the use of specialized APIs causes vendor lock-in. However, the performance loss that still exists in full virtualization make para- or hybrid para- and full virtualization technologies attractive despite the potential lock-in. It is also argued that as hardware vendors are developing support for full virtualization in hardware, these technologies are able to compete with para-virtualization with regard to performance.

The main problem that interviewees see with virtualization is that it may be used for the wrong tasks. There is a strong opinion that products should be able to work independently, without virtualization. Virtualization may later be added to achieve properties such as consolidation and isolation. However, it is argued that virtualization should not be used to introduce fault tolerance, as this is often better solved in the

application layer where the semantics of the task at hand is better understood.

Although people involved in product development strive towards using more Common Off-The-Shelf (COTS) hardware, some specific tasks are still handled by specialized hardware such as Digital Signal Processors (DSPs). This may lead to problems when using virtualization, as the hardware becomes abstracted from the application.

When it is decided that a project should utilize virtualization, some tests of the alternatives are typically conducted during a feasibility study. As a lot of alternatives exist, the time required for this research can be longer than what is allocated. This may lead to poorly supported decisions and favorisation through familiarity, and the most suitable technology might hence not be chosen.

Conclusions from the Interviews

As mentioned in the summary of the interviews, concerns were raised about the appropriateness of para-virtualization as it might create unnecessary vendor lock-in as virtualization support in hardware becomes more common. The hardware support might however not always be a suitable solution when e.g. dealing with shared hardware such as partitioning up a hard drive to share it between VMs, which is very difficult to totally virtualize in hardware. Also, the risk for vendor lock-in to a certain hardware vendor is something not to be overlooked. As long as Linux vendors like SuSE or Red Hat only support one virtualization technology, a change of VMM might probably also result in a change in Linux vendor. Today, VMware is the only large stand-alone VMM vendor.

The question whether or not virtualization is the right choice to solve a certain problem at hand is neither simple nor unambiguous. A simplified view of this is that virtualization is something that should not be used to solve problems, but rather to introduce new features like consolidating the functions of a finished product onto fewer hardware elements. While this is a rather harsh view, it might be beneficial if it discourages developers from making virtualization a panacea, when problems are better solved by other means.

When talking about software, security and function are often two competing aspects. If the goal is a highly secure VMM, it will probably be anything but feature-rich. As different VMMs has different target applications, the security requirements may instead be answered through e.g. certification programs such as EAL 6 [81], that demand that the functionality of all the code is proven to be correct.

3.1.2 Selecting Technologies to Test

Simply put, the technologies that were available for deployment using only Linux, and preferably had some sort of commercial backing were selected for testing. This means that technologies like Microsoft's Hyper-V were skipped, as it requires a Windows 2008 Server installation. The full list of tested technologies is found in Table 3.1. Benchmarking results for VMware products is not presented, as the written consent of VMware inc. is required to publish such figures.

3.1.3 Design and Implementation of the Benchmarking Tool

The benchmarking tool should, as mentioned, require as little human interaction as possible in order to add new virtualization technologies for testing during one feasibility study. The benchmarking tool is designed to be future proof, that is, have a dynamic and modular design that is also easily extendable. The design of the tool and the APIs,

Table 3.1: List of virtualization technologies with commercial support. Version numbers in parenthesis are Linux kernel versions used.

Technology	Vendor	Version used
Xen	XenSource, Citrix, Novell	3.3.1 (2.6.27)
KVM	Red Hat	No separate version number (2.6.27)
OpenVZ	Parallels	1.43 (2.6.27.11)
Linux-VServer	(Community)	2.3.0.36.4 (2.6.27.14)

is shown in Figure 3.1. Classes exist that represent and implement access methods to the hypervisor, the benchmark, the host and the load generator. The classes representing the real implementations inherit the properties of these classes. As for hypervisors, an intermediate libvirt class is introduced, that implements the libvirt interface to simplify code reuse when a hypervisor implements the libvirt API. The basic operation of the tool is described in Figure 3.2. After dynamically loading and initiating each enabled hypervisor, the enabled benchmarks is run on said hypervisor. If a load generator is enabled, the benchmark initialize and run the load generators.

As the benchmarking tool connects to VMs using SSH, and requires much string handling and shell interacting, the scripting language Perl is used to implement the tool. In Perl, both settings and schemata are most easily represented using hash references, as this is the preferred input format to existing configuration parsers and validators.

Since it is not always interesting to run all load generators on all benchmarks, the initialization and running of load generators is located in the `run` method of the Benchmark class. This allows for the Benchmark settings object to define what load generators that should be run during the benchmarking. As a default, no load generators are loaded. The separation of the `init`, `run`, and `finalize` methods for the benchmarks and load balancer is done in order to enable the non-benchmarking methods to be run in parallel.

3.1.4 Selecting Benchmarks

The main reason for creating the benchmarking tool is that the work may easily be replicated in the future (both for validation and to re-run the tests on newer hardware), and to enable replacement of the tests, which is useful e.g. when searching for a virtualization technology that fulfil requirements other than the requirements explored in this thesis. Because of this, a large variety of tests should be available. However, packaging tests do take time, and it is not possible to package every slightly interesting test. Hence, as few tests as possible from each performance profile were chosen, weighing in the similarity to other tests from the same area, the general popularity of the test, and the estimated future interest of the test.

For example, I/O testing with regard to networking is often done either in a general way, such as pure bandwidth tests, or alternatively as a highly application-specific task, such as measuring the time required to transfer a well-defined set of packets. As the latter test is only of interest for one application of virtualization, it is not prioritized to package such tests for the benchmarking tool at this point. As for general network I/O testing, most tools work very similarly and therefore only one needs to be packaged. For disk I/O, there are a number of general tests for very different aspects of I/O performance, e.g. bandwidth, latency, and small file operations. Because of this, a few disk benchmarks are selected. A list of the packaged benchmarks is found in Table 3.2.

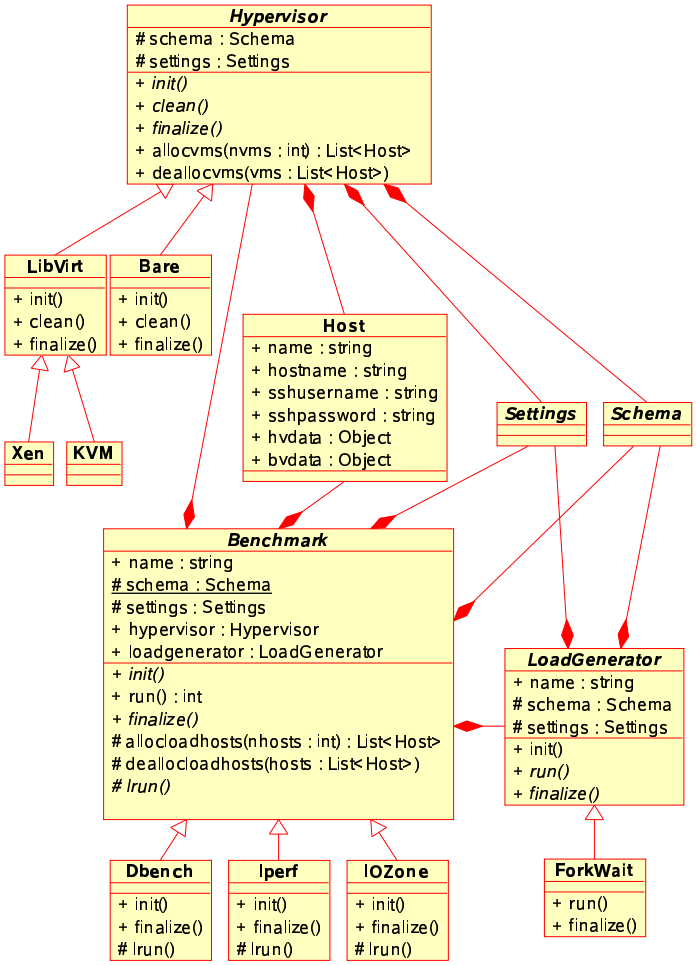

Figure 3.1: UML diagram of the benchmarking tool.

3.2 Benchmarking

A technology preview board with 6 GB of RAM and a 4 core 2.4 GHz Intel processor based on the *Nehalem* microarchitecture was allocated for the testing of technologies and the development of the benchmarking tool. This processor features the latest version of the Intel VT-x virtualization extensions (described in Section 2.4) and supports Hyper Threading. It is likely that future Ericsson technology will incorporate a processor with a similar microarchitecture. In addition to the board, a HP laptop with a Gigabit Ethernet port was used for network I/O benchmarking and as test controller. The VMs created for the test are based on Ubuntu Linux 8.10 server and were allocated one processor core and 384 MB of RAM each.

The outline of the benchmarking strategy is to first verify that CPU-bound problems are possible to virtualize without significant performance impact. Next, the scalability of different technologies is investigated. Finally, an example of a more application-specific series of tests are performed, where different configurations of the Xen virtualization technology are tested.

```
for all h in hypervisors do
  h.init()
  for all b in benchmarks do
    b.init()
    b.run()
    b.finalize()
  end for
  h.finalize()
end for
define function b.run() as
  for all l in loadGenerators do
    l.init()
    l.run()
    b.lrun()
    l.finalize()
  end for
end function
```

Figure 3.2: The basic operation of the benchmarking tool.

Table 3.2: List of benchmarks packaged with the benchmarking tool.

Benchmark	Performance characteristic
Bonnie	Disk I/O
DBench	Simulated file server (disk I/O)
dd	Disk I/O
Dhrystone	Integer performance
FBench	Floating point performance
IOZone	Disk and cache I/O
iperf	Network I/O
Linux kernel compile	Disk I/O and integer operations
Linux kernel compile with regard to scalability	Disk I/O and integer operations
LMbench2	Syscall benchmarking
Mstone	Media-centric Erlang benchmark
rsync	Disk and network I/O
SysBench	Database benchmark
SysBench with regard to scalability	Database benchmark

3.2.1 CPU-bound benchmarks

As the hardware in use feature virtualization extensions, the performance of CPU-bound problems should be close to that achieved when running the same problems on native hardware. As noted earlier, the very definition of virtualization include the premise that as much code as possible should be run naively on the processor. In order to verify this, which is needed before concentrating on other aspects of benchmarking, two user space CPU-bound programs are run: FBench, a floating point arithmetic benchmark that performs a specified number of iterations, in this case 20.000, and Dhrystone, an integer arithmetic benchmark. Dhrystone is run in three different settings, the first two being with and without an optional register optimization that ensures that often used variables are placed in processor registers. The third run uses this register option, as well as one level of compiler optimization (*-O1*). As Figure 3.3 show, the difference when the benchmark runs in a VM is very small, being at a maximum 3% slower for FBench and less than 1% to 3% for Dhrystone.

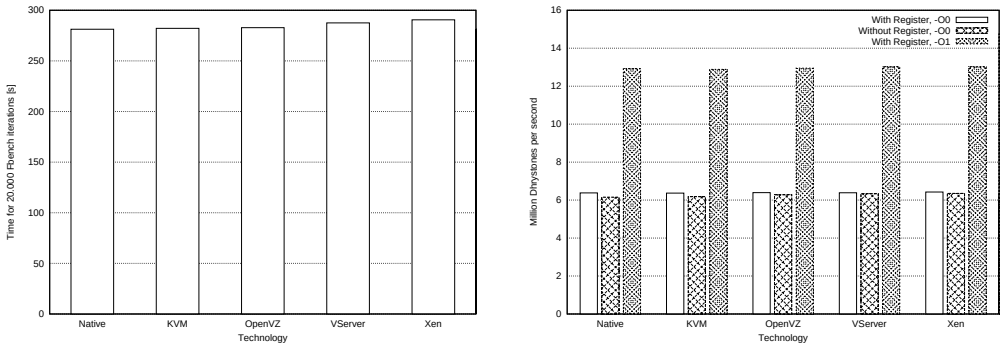

Figure 3.3: As anticipated, there is a small to no performance impact when running a CPU-bound problem on a virtualization layer.

3.2.2 Scalability

When measuring scalability, non-scalability specific benchmarks are used. These benchmarks are run in parallel, both with and without a VMM, to determine how this affects scheduling of said benchmarks. Preferably, the benchmark can be summarized as e.g. the total throughput of transactions in SysBench. In this case, KVM lacks the reliable high precision time SysBench require, and because of this, a number of parallel Linux kernel compilation runs are used to measure the scalability of a technology. This method of measuring scalability is used by e.g. Camargos et al [43]. As the results in Figure 3.4 show, all technologies except for Xen perform very well. When using 3 VMs, the time registered for Xen is only 9% higher than that of native hardware. When using 8 VMs, Xen show the most overhead by using 173% more time than without a VMM, while KVM only show an increased compile time by 29%. As pictured, the scalability of Linux-VServer and OpenVZ is slightly better than when running on a stock kernel. This can be explained by the fact that both technologies incorporates their own scheduling in the OS kernel to ensure fair scheduling among containers, which the used benchmark favors to a certain level.

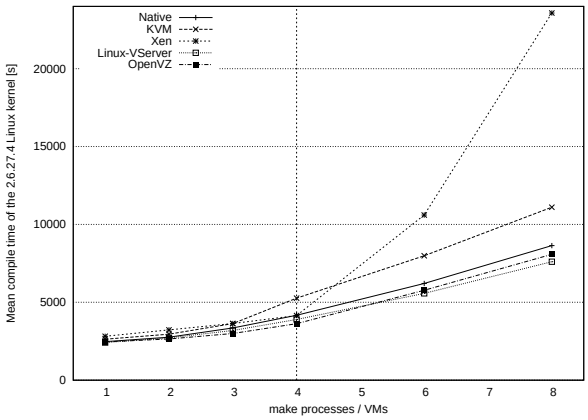

Figure 3.4: Compile time of version 2.6.27.4 of the Linux kernel with and without different VMMs. A vertical bar is placed where the number of make processes equals the number of processor cores.

As specified, the system used to run the benchmarks feature a 4 core processor, and as Figure 3.4 show, the compile times in the Xen case start to increase substantially when using more than 4 VMs. It would be interesting to examine the reason for the fast increase of compile time for the case of an over scheduled Xen system, as no other documented problems of this nature is to be found. Scalability testing of Xen is done by e.g. Theurer et al [85], but without over scheduling of the processing elements available. One possible reason for this might be the cache effects explored more in the application test in Section 3.2.3, as the Xen VMs in this case use image files as disk back-end.

3.2.3 Benchmarking of Deployment Options

When a virtualization technology is chosen, various details regarding the configuration of said technology have to be chosen as well. Benchmarking the different configuration alternatives can prove helpful when deciding on this configuration. Consider a scenario where the product is to use Xen as the virtualization technology, and benchmarking should investigate whether para-virtualization can be avoided, as the product to be virtualized is already released, and if each VM should be allocated a hard drive partition, or if it is sufficient to use image files as disk back-ends.

As the VMs are to run media applications, the Mstone benchmark is used for application-specific performance. In addition, I/O is tested using Bonnie, IOZone, and dd, all benchmarks for large, sequential disk I/O; DBench, that emulates a file server; and iperf that tests network I/O.

Testing the need for para-virtualization is done by creating three setups. These are para-virtualized, fully virtualized and fully virtualized with para-virtualized I/O drivers for disk and network, referred to in the figures as *Full*, *Para* and *Hybrid* respectively. Testing performance in the case of image files and partitions as disk back-ends is done by setting up a VM of each technique to use files placed on an ext3 file system as well as partitions as disk back-ends. These cases sum up to six setups that are tested.

The first run of the disk I/O benchmarks exposed some problems with the setup, as shown in Figure 3.5. Before the tests started, it was expected that the partition-

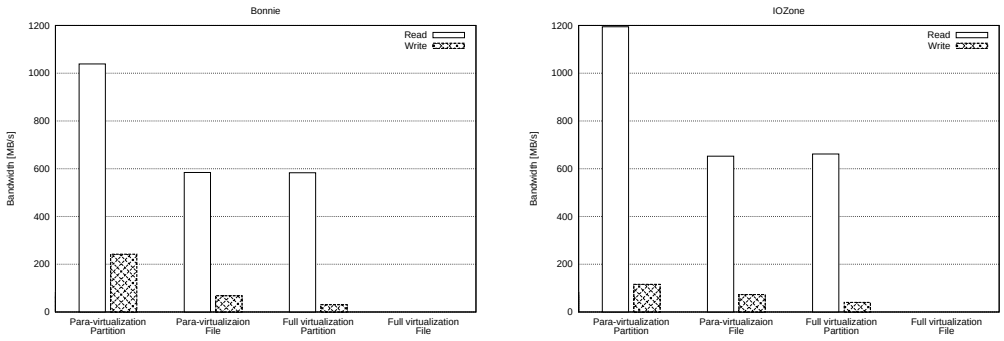

Figure 3.5: Erroneous benchmarking results due to cache effects.

based para-virtualized alternative would result in higher I/O bandwidth than the other alternatives due to the use of more efficient para-virtualized drivers and the smaller overhead of only having one file system between the application and the physical device. This proved not to be the case due to the caching of both reads and writes in the emulated hardware device and in the file system of the host which lead to para-virtualization using a partition back-end summing the speed to around 86.5 MB/s for reading and around 71 MB/s for writing while using a file back-end summed to around 1 GB/s for reading and around 120 MB/s for writing. As the connected disk is using a 300 MB/s SATA interface, these numbers proved to be obviously false. While the numbers produced may help to identify and analyze the different levels of cache present, the most interesting case is reading of a non-cached file to determine the bulk transfer speed from hardware. Additionally, in this particular application case, it is required that no writes are cached as the disk is to be used as persistent storage.

To solve this, Xen developers were contacted through the support organization. According to the information obtained, it is possible to disable the cache in the hardware emulation layer, but this option is not exposed in the management tools of Xen and is therefore not supported. To solve this, the file to be written and read have to be larger than the size of the memory available for caching. This is achieved by removing 4 GB of physical RAM, leaving 2 GB, disabling swap and adjusting the working set to 2 GB instead of 1 GB where possible. The results seen in Figure 3.6 confirm the expected results when using the Bonnie, IOZone and dd benchmarks. Note however that it cannot be expected that the figures obtained from these benchmarks are completely free of cache effects, as the cache is not explicitly turned off, only thrashed. The DBench benchmark, also represented in Figure 3.6, still show signs of cache effects. This is because the working set of this benchmark is not configurable, and may be used to show how thorough a benchmark design must be. As Bonnie, IOZone and dd all show similar numbers, confirming an increase of 80-100% in I/O bandwidth when shifting from a file back-end to a partition back-end, it can be assumed that the bandwidths obtained are relatively sound although some values measured are faster than native speed. The Mstone and iperf benchmarks, shown in Figure 3.7, are not dependent on the disk back-end, and thus give identical results for file and partition. Despite this, the benchmarking diagrams are still divided by file and partition like the diagrams in Figure 3.6 for clarity. It can be noted that the Mstone result is a bit lower on native hardware. The cause of

this is probably due to the fact that Mstone had to be run with only one scheduler in order to get a comparable result in the native case.

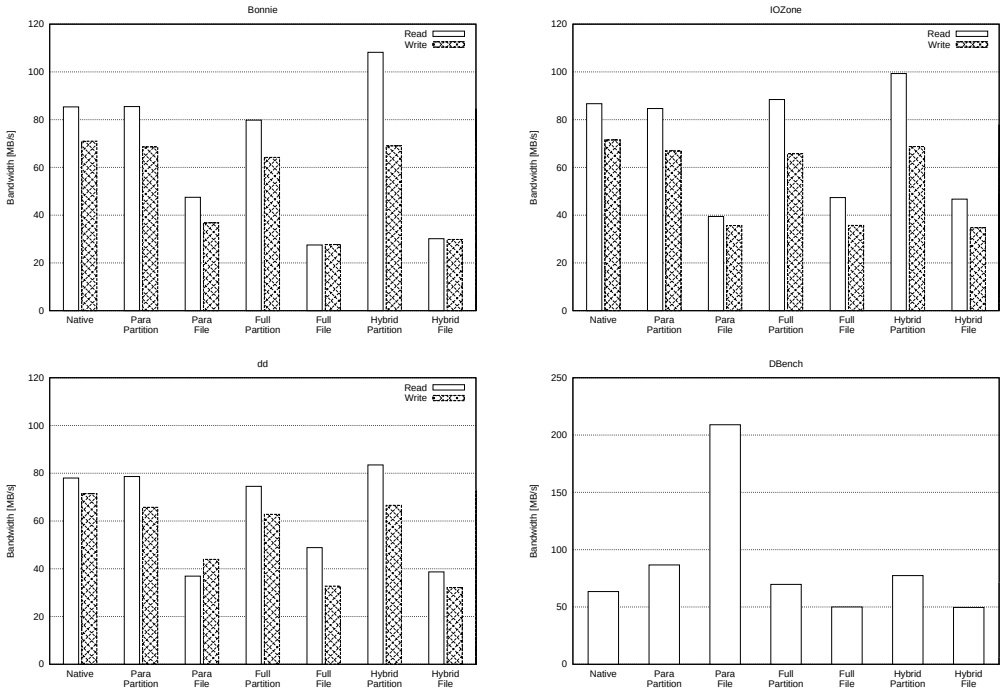

Figure 3.6: Disk speeds benchmarked using Bonnie, IOZone, dd and DBench. DBench still suffer from cache effects.

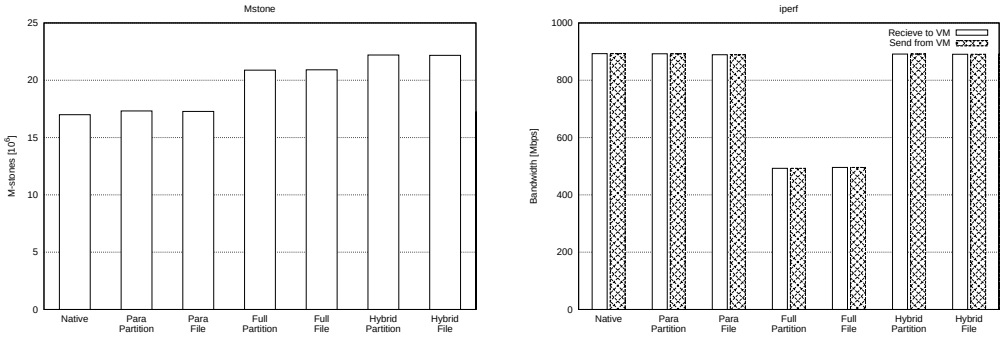

Figure 3.7: Both the Mstone and the iperf benchmark are unaffected by the disk backend.

Chapter 4

Conclusions

As for the ultimate virtualization technology, no technology tested suits all applications. Especially, it is not certain that there will be only one technology for any foreseeable future. As noted, KVM is quickly moving forward as it is backed by Red Hat, who employs an impressive amount of developers. The fact that para-virtualized technologies like Xen are looking at simplifying the implementations by using hardware support for virtualization [55], and the fact that KVM support para-virtualized hardware drivers indicate that Xen and KVM will converge somewhat, technology-wise.

When choosing a full or para-virtualized technology, Xen have the upper hand due to its large community, commercial backing and support organization. However, as noted during the scalability tests, Xen performs poorly when over scheduled and suffer from a caching effect when using fully virtualized drivers. This caching effect may be acceptable when running stateless applications, but when transactional properties are required, fully virtualized drivers that use this caching are unacceptable. These problems show that in order to make a correct choice of virtualization technology and in tuning the deployment options, a lot of research must be done to understand the underlying requirements and technologies that are to be tested. Before dismissing Xen in an over scheduled environment, a new set of test should be run to determine if the scalability problems were due to the virtual drivers used.

As mentioned, OS-level virtualization is not virtualization in the traditional sense. As all containers share one kernel, there is less overhead and fair scheduling may be simpler to provide. However, the sharing of one kernel may hinder in a complex environment, where for example different customers have different requirements. The ability to create a custom environment where a customer has full control is in most cases harder when using OS-level virtualization. When discussing OpenVZ and Linux-VServer, the performance of each solution is very similar, which means that other aspects needs to be explored. For example, OpenVZ support libvirt for management and CIM through libvirt-CIM.

The benchmarking performed show that the overhead of virtualization can be impressively low, even for a young technology like KVM. The observed problems are most often isolated to overhead in memory usage and I/O operations. A number of solutions are identified that increase performance when using I/O, such as the usage of para-virtualized drivers, either in a totally para-virtualized environment or as a complement to a fully virtualized environment, and for the disk I/O case to use partitions or disks private to a VM as opposed to image files stored on a file system.

When discussing management of VMMs and VMs, the choice is between CIM and libvirt, as neither SNMP nor NETCONF are supported by any VMM, with an exception for read only data being available through SNMP for Xen and VMware. By looking at which VMM that support which technology, it is clear that the VMMs that support libvirt (Xen, KVM, OpenVZ, LXC, UML, QEMU [1]) are all projects with deep roots in the open source community, while technologies that support CIM (e.g. Xen, VMware) are commercialized products that target large-scale installations. This obvious split in interests and protocols may however change shortly, as Red Hat is a supplier of enterprise solutions. If Red Hat continues to push KVM and libvirt, libvirt might be actualized as a commercial management solution. To conclude, libvirt and CIM can solve the same problems in somewhat different ways. While CIM may have the advantage of delivering a more complete solution where physical machines, VMMs and VMs are presented as a whole, libvirt have a smaller footprint and a simpler and more virtualization-centric design.

4.1 Future work

While it would fun be to cover all known virtualization technologies and all benchmarks regardless of their respective significance in order to determine the best virtualization technology for every use case, some limit had to be imposed. For testing, only the larger virtualization technologies with commercial backing were chosen, and benchmarks were restricted to the more general tests. A more complete future study may include both commercial and community technologies to get a more complete view of the available technologies. As mentioned, e.g. VMware ESX is avoided due to the VMware end user license agreement that forbid benchmarking data to be published.

The software developed, i.e. the benchmarking tool, can be improved with e.g. a unified presentation format, the possibility to repeat tests to calculate means and deviations, support for finer grained configuration, i.e. different benchmark options per VMM, and better cleaning to avoid the reboot of virtual machines between benchmarks. If the design proves to be able to fulfill the requirements that exist for a benchmarking tool, it will hopefully be used in further testing efforts.

The SPECvirt_sc2009, as described in Section 2.7.1, will mostly test for general performance rather than some specialized usage cases. However, it will certainly be commonly used when testing different virtualization technologies in the future. As for specialized benchmarks, the tool written in this thesis will hopefully be used and extended when specialized tests are being performed at Ericsson.

Chapter 5

Acknowledgements

First of, a big thank you to the internal supervisor Johan Tordsson and the two external supervisors Fredrik Svahn and Tom Rindborg. It is pretty obvious that this thesis would not have been completed without the help of these people.

I would also like to thank the interviewees for taking time from their schedule to meet with me and answer my questions: Jonas Bjurel, Hans Brolin, Thomas Johansson, Lars-Örjan Kling, Johan Kölhi, Mats Ljunggren, Henrik Persson, Peter Svensson, and Staffan Ålund. Out of these, special mentions are in order for Johan Kölhi, for all interesting discussions and for the invitation to speak on an Ericsson virtualization day, and for Thomas Johansson considering the help provided when dealing with VMware.

I also want to acknowledge the whole Linux Development Center at Ericsson, where the time writing this thesis has been spent. Special mentions are in order for my lunch mates, Kerstin Jonsson and Mikael Nyström.

I am also thankful for having two friends that probably have something to do with my being here and doing this, Malin and Tomas Pihl.

Lastly, my friends and family (no one mentioned, no one forgotten) deserve a great deal of gratitude for just about everything.

References

[1] libvirt: The virtualization API. Web page, December 8, 2008. http://libvirt.org/.

[2] libvirt: XML Format. Web page, December 12, 2008. http://www.libvirt.org/format.html.

[3] The virtualization CIM API. Web page, December 11, 2008. http://libvirt.org/CIM/.

[4] K. Adams and O. Agesen. A comparison of software and hardware techniques for x86 virtualization. *Proc. of the 12th international conference on Architectural support for programming languages and operating systems*, pages 2–13, 2006.

[5] Advanced Micro Devices, Inc. AMD Unveils Virtualization Platform. Web page, December 12, 2008. http://www.amd.com/us-en/Weblets/0,,7832_8366_7595~96162,00.html.

[6] Z. Amsden, D. Arai, D. Hecht, A. Holler, and P. Subrahmanyam. VMI: An Interface for Paravirtualization. In *Proc. of the 2006 Linux Symposium*, pages 363–378, 2006.

[7] C. Augier. Real-Time Scheduling in a Virtual Machine Environment. *Junior Researcher Workshop on Real-Time Computing 2008*.

[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[9] F. Bellard. QEMU - About. Web page, December 12, 2008. http://bellard.org/qemu/about.html.

[10] F. Bellard. QEMU - Status. Web page, December 12, 2008. http://bellard.org/qemu/status.html.

[11] M. Bjorklund. YANG - A data modeling language for NETCONF. Internet-Draft draft-ietf-netmod-yang-02, Internet Engineering Task Force, November 2008. Work in progress.

[12] U. Blumenthal and B. Wijnen. User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3). RFC 3414 (Standard), December 2002.

[13] J.P. Casazza, M. Greenfield, and K. Shi. Redefining Server Performance Characterization for Virtualization Benchmarking. *Intel technology Journal*, 10(03).

[14] J. Case, D. Harrington, R. Presuhn, and B. Wijnen. Message Processing and Dispatching for the Simple Network Management Protocol (SNMP). RFC 3412 (Standard), December 2002.

[15] J.D. Case, M. Fedor, M.L. Schoffstall, and J. Davin. Simple Network Management Protocol. RFC 1067, August 1988. Obsoleted by RFC 1098.

[16] I. P. Christian. Monitoring Xen via SNMP. Web page, December 11, 2008. `http://pookey.co.uk/blog/archives/52-Monitoring-Xen-via-SNMP.html`.

[17] Citrix Systems, Inc. Xen paravirt_ops for x86 Linux. Web page, December 30, 2008. `http://wiki.xensource.com/xenwiki/XenParavirtOps`.

[18] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *Proc. of the 2nd conference on Symposium on Networked Systems Design & Implementation*, volume 2, pages 273–286, 2005.

[19] R.J. Creasy. The Origin of the VM/370 Time-Sharing System. *IBM Journal of Research and Development*, 25(5):483–490, 1981.

[20] D. Smith et al. Linux Containers. Web page, December 1, 2008. `http://lxc.sourceforge.net/`.

[21] T. Deshane, Z. Shepherd, J.N. Matthews, M. Ben-Yehuda, A. Shah, and B. Rao. Quantitative Comparison of Xen and KVM. *Proc. of Xen Summit Boston*, 2008.

[22] J. Dike. A user-mode port of the linux kernel. In *Proc. of the 4th conference on 4th Annual Linux Showcase & Conference, Atlanta*, pages 7–7, 2000.

[23] Distributed Management Task Force, Inc. Common Information Model (CIM) Standards. Web page, December 8, 2008. `http://www.dmtf.org/standards/cim/`.

[24] Distributed Management Task Force, Inc. Virtualization Management (VMAN) Initiative. Web page, December 8, 2008. `http://www.dmtf.org/standards/mgmt/vman/`.

[25] Distributed Management Task Force, Inc. WBEM: CIM-XML. Web page, December 8, 2008. `http://www.dmtf.org/standards/wbem/CIM-XML/`.

[26] Distributed Management Task Force, Inc. Web-Based Enterprise Management (WBEM). Web page, December 8, 2008. `http://www.dmtf.org/standards/wbem/`.

[27] Distributed Management Task Force, Inc. Web Services for Management. Web page, December 8, 2008. `http://www.dmtf.org/standards/wsman/`.

[28] K. Džonsons. Logical Resource Isolation in the NetBSD Kernel. In *Proc. of AsiaBSDCon*, 2008.

[29] P. Emelianov, D. Lunev, and K. Korotaev. Resource Management: Beancounters. In *Proc. of the 2007 Linux Symposium*, pages 285–297, 2007.

[30] R. Enns. NETCONF Configuration Protocol. RFC 4741 (Proposed Standard), December 2006.

[31] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson. Safe hardware access with the Xen virtual machine monitor. In *OASIS ASPLOS 2004 workshop*, 2004.

[32] Free Software Foundation, Inc. GDB: The GNU Project Debugger. Web page, September 23, 2008. `http://www.gnu.org/software/gdb/`.

[33] R. Frye, D. Levi, S. Routhier, and B. Wijnen. Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework. RFC 3584 (Best Current Practice), August 2003.

[34] T. Goddard. Using NETCONF over the Simple Object Access Protocol (SOAP). RFC 4743 (Proposed Standard), December 2006.

[35] S. Hand, A. Warfield, K. Fraser, E. Kotsovinos, and D. Magenheimer. Are Virtual Machine Monitors Microkernels Done Right? *ACM SIGOPS Operating Systems Review*, 4(1):95–99, 2006.

[36] D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. RFC 3411 (Standard), December 2002. Updated by RFC 5343.

[37] IETF Secretariat. Network Time Protocol (ntp) Charter. Web page, December 30, 2008. `http://www.ietf.org/html.charters/ntp-charter.html`.

[38] Intel Corporation. Intel Delivers New Era For Virtualization. Web page, December 12, 2008. `http://www.intel.com/pressroom/archive/releases/-20051114comp.htm`.

[39] S.T. King, G.W. Dunlap, and P.M. Chen. Operating system support for virtual machines. In *Proc. of the USENIX Annual Technical Conference 2003 on USENIX Annual Technical Conference table of contents*, pages 6–6, 2003.

[40] S.T. King, G.W. Dunlap, and P.M. Chen. Debugging operating systems with time-traveling virtual machines. In *Proc. of the USENIX 2005 Annual Technical Conference*, 2005.

[41] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the Linux Virtual Machine Monitor. In *Proc. of the 2007 Linux Symposium*, 2007.

[42] Ksplice, Inc. Ksplice: Rebootless Linux kernel upgrades. Web page, December 1, 2008. `http://www.ksplice.com/`.

[43] F. Laudares Camargos, G. Girdard, and B. des Ligneris. Virtualization of Linux servers: a comparative study. In *Proc. of the 2008 Linux Symposium*, volume 1, pages 63–76, 2008.

[44] E. Lear and K. Crozier. Using the NETCONF Protocol over the Blocks Extensible Exchange Protocol (BEEP). RFC 4744 (Proposed Standard), December 2006.

[45] D. Levi, P. Meyer, and B. Stewart. Simple Network Management Protocol (SNMP) Applications. RFC 3413 (Standard), December 2002.

[46] Linux Kernel Organization, Inc. ChangeLog-2.6.20. Web page, December 12, 2008. `http://kernel.org/pub/linux/kernel/v2.6/ChangeLog-2.6.20`.

[47] Linux Kernel Organization, Inc. The -mm patches to the Linux kernel. Web page, December 1, 2008. http://www.kernel.org/patchtypes/mm.html.

[48] Linux-VServer Wiki Team and Public Relations Team. Paper - Linux-VServer. Web page, December 1, 2008. http://linux-vserver.org/Paper.

[49] V. Makhija, B. Herndon, P. Smith, L. Roderick, E. Zamost, and J. Anderson. VMmark: A scalable benchmark for virtualized systems. Technical report, VMWare, 2006.

[50] K. McCloghrie and M.T. Rose. Management Information Base for network management of TCP/IP-based internets. RFC 1066, August 1988. Obsoleted by RFC 1156.

[51] K. McCloghrie and M.T. Rose. Structure and identification of management information for TCP/IP-based internets. RFC 1065 (Standard), August 1988. Obsoleted by RFC 1155.

[52] A. Menon, A.L. Cox, and W. Zwaenepoel. Optimizing Network Virtualization in Xen. In *Proc. USENIX Annual Technical Conference (USENIX 2006)*, pages 15–28, 2006.

[53] meyering@redhat.com. git.et.redhat.com Git - libvirt.git/tags. Web page, December 12, 2008. http://git.et.redhat.com/?p=libvirt.git;a=tags.

[54] J. Nakajima, A. Mallick, I. Pratt, and K. Fraser. X86-64 XenLinux: Architecture, Implementation, and Optimizations. In *Proc. of the 2006 Linux Symposium*, 2006.

[55] J. Nakajima and A.K. Mallick. Hybrid-Virtualization - Enhanced Virtualization for Linux. In *Proc. of the 2007 Linux Symposium*, 2007.

[56] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig. Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization. *Intel Technology Journal*, 10(3):167–177, 2006.

[57] Novell, Inc. SUSE Linux Enterprise Server 10 Service Pack 2 for AMD64 & Intel EM64T: Package Descriptions. Web page, December 12, 2008. http://www.novell.com/products/linuxpackages/server10/sp2/-x86_64/index_group.html.

[58] G.J. Popek and R.P. Goldberg. Formal Requirements for Virtualizable Third Generation Architectures. *Communications of the ACM*, 17(7):412–421, 1974.

[59] I. Pratt, K. Fraser, S. Hand, C. Limpach, A. Warfield, D. Magenheimer, J. Nakajima, and A. Malick. Xen 3.0 and the art of virtualization. In *Proc. of the 2005 Linux Symposium*, 2005.

[60] R. Presuhn. Management Information Base (MIB) for the Simple Network Management Protocol (SNMP). RFC 3418 (Standard), December 2002.

[61] R. Presuhn. Transport Mappings for the Simple Network Management Protocol (SNMP). RFC 3417 (Standard), December 2002. Updated by RFC 4789.

[62] R. Presuhn. Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP). RFC 3416 (Standard), December 2002.

[63] Qumranet inc. Status - Kernel based Virtual Machine. Web page, December 12, 2008. http://kvm.qumranet.com/kvmwiki/Status.

[64] Qumranet inc. Status - Kernel based Virtual Machine. Web page, January 23, 2009. http://kvm.qumranet.com/kvmwiki/Paravirtualized_networking.

[65] Qumranet inc. Status - Kernel based Virtual Machine. Web page, January 23, 2009. http://kvm.qumranet.com/kvmwiki/Paravirtualized_block_device.

[66] Qumranet inc. Status - Kernel based Virtual Machine. Web page, January 23, 2009. http://uml.jfdi.org/uml/Wiki.jsp?page=NamingConventionForUMLPatchedKernels.

[67] Red Hat, inc. The Evolution of Virtualization: Qumranet joins Red Hat. Web page, December 12, 2008. http://www.redhat.com/promo/qumranet.

[68] Red Hat, Inc. Virtualization Guide. Web page, December 12, 2008. http://www.redhat.com/docs/manuals/enterprise/RHEL-5-manual/-Virtualization-en-US/index.html.

[69] RimuHosting. RimuHosting Bliki :: knowledgebase/linux/xen. Web page, December 1, 2008. http://bliki.rimuhosting.com/space/knowledgebase/linux/xen.

[70] J.S. Robin and C.E. Irvine. Analysis of the Intel Pentium's ability to support a secure virtual machine monitor. *Proc. of the 9th conference on USENIX Security Symposium*, pages 10–10, 2000.

[71] R. Russell. Lguest: Frequently Asked Questions. Web page, December 1, 2008. http://lguest.ozlabs.org/faq.html.

[72] R. Russell. Lguest: The Simple x86 Hypervisor. Web page, December 1, 2008. http://lguest.ozlabs.org.

[73] N. Salmoria and the MAME team. Multiple Arcade Machine Emulator. Web page, December 12, 2008. http://www.mamedev.org/.

[74] J. Smith and R. Nair. *Virtual Machines: Versatile Platforms for Systems and Processes (The Morgan Kaufmann Series in Computer Architecture and Design)*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2005.

[75] Standard Performance Evaluation Corporation. SPEC Virtualization Committee. Web page, October 10, 2008. http://www.spec.org/specvirtualization/.

[76] Sun Microsystems, Inc. FAQs for Sun xVM VirtualBox. Web page, December 12, 2008. http://www.sun.com/software/products/virtualbox/faqs.jsp.

[77] Sun Microsystems, Inc. Sun Virtual Desktop Infrastructure Software. Web page, December 12, 2008. http://www.sun.com/software/vdi/index.jsp.

[78] Sun Microsystems, Inc. Sun xVM Ops Center. Web page, December 14, 2008. http://www.sun.com/software/products/xvmopscenter/index.jsp.

[79] Sun Microsystems, Inc. xVM Server Introduction. Web page, December 14, 2008. http://www.xvmserver.org/about.html.

[80] Sun Microsystems, Inc. Zones and Containers FAQ. Web page, December 14, 2008. http://www.opensolaris.org/os/community/zones/faq/.

[81] The Common Criteria Portal. Official CC/CEM versions - The Common Criteria Portal. Web page, December 30, 2008. http://www.commoncriteriaportal.org/-contact.html.

[82] The Open Group. *Systems Management: Common Manageability Programming Interface (CMPI)*, December 11, 2008. http://www.opengroup.org/bookstore/-catalog/c051.htm.

[83] The OpenVZ project. FAQ. Web page, December 12, 2008. http://wiki.openvz.org/FAQ.

[84] A. Theurer, K. Rister, and S. Dobbelstein. A Survey of Virtualization Workloads. In *Proc. of the 2008 Linux Symposium*, volume 2, pages 215–225, 2008.

[85] A. Theurer, K. Rister, O. Krieger, R. Harper, and S. Dobbelstein. Virtual Scalability: Charting the Performance of Linux in a Virtual World. In *Proc. of the 2006 Linux Symposium*, 2006.

[86] VMware, Inc. Features of ESX - Production-Proven Hypervisor. Web page, December 13, 2008. http://www.vmware.com/products/vi/esx/esx_features.html.

[87] VMware, Inc. Installing the ESX Server SNMP Agents. Web page, December 11, 2008. http://www.vmware.com/support/esx21/doc/esx21admin_snmpagents.html.

[88] VMware, Inc. VMware ESXi FAQs. Web page, December 13, 2008. http://www.vmware.com/products/esxi/uses.html.

[89] VMware, Inc. VMware Workstation. Web page, December 13, 2008. http://www.vmware.com/products/ws/.

[90] M. Wasserman and T. Goddard. Using the NETCONF Configuration Protocol over Secure SHell (SSH). RFC 4742 (Proposed Standard), December 2006.

[91] A. Whitaker, R.S. Cox, and S.D. Gribble. Configuration Debugging as Search: Finding the Needle in the Haystack. *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, 6:6–6, 2004.

[92] B. Wijnen, R. Presuhn, and K. McCloghrie. View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP). RFC 3415 (Standard), December 2002.

[93] Wind River Systems, Inc. Using a Hypervisor to Manage Multi-OS Systems. Presentation, December 1, 2008. http://2008ftf.ccidnet.com/pdf/PN202.pdf.

[94] XenSource inc. XenFaq. Web page, December 12, 2008. http://-wiki.xensource.com/xenwiki/XenFaq.

[95] A. Zeichick. Processor-Based Virtualization, AMD64 Style, Part I. Web page, December 12, 2008. http://developer.amd.com/documentation/articles/pages/-630200614.aspx.

[96] A. Zeichick. Processor-Based Virtualization, AMD64 Style, Part II. Web page, December 12, 2008. http://developer.amd.com/documentation/articles/pages/-630200615.aspx.

[97] E. Zhai, G.D. Cummings, and Z. Dong. Live Migration with Pass-through Device for Linux VM. In *Proc. of the 2008 Linux Symposium*, volume 2, pages 261–267, 2008.

Appendix A

List of Acronyms

API Application Programming Interface

BEEP Blocks Extensible Exchange Protocol

CIM Common Information Model

CIMOM CIM Object Manager

CMPI Common Management Programming Interface

CMS Conversational Monitor System

COTS Common Off-The-Shelf

CP Control Program

CTSS Compatible Time-Sharing System

DMA Direct Memory Access

DMTF Distributed Management Task Force

DSP Digital Signal Processor

GPOS General Purpose OS

HIDS Host-based Intrusion Detection System

HVM Hybrid Virtual Machine

IOMMU I/O Memory Management Unit

IPC Inter-Process Communication

KVM Kernel-based Virtual Machine

LXC Linux Containers

MAME Multiple Arcade Machine Emulator

MIB Management Information Base

MOF Managed Object Format

MOF Meta-Object Facility

NAS Network Attached Storage

NETCONF Network Configuration Protocol

NTP Network Time Protocol

OMG Object Management Group

OS Operating System

OVF Open Virtualization Format

RFC Request For Comments

RSCS Remote Spooling and Communications Subsystem

RTOS Real-Time OS

SAN Storage Area Network

SSH Secure Shell

SSL Secure Socket Layer

SI Scheduler Interface

SMP Symmetric Multiprocessing

SNMP Simple Network Management Protocol

SNMPv1 SNMP version 1

SNMPv3 SNMP version 3

SOAP Simple Object Access Protocol

SVD Scheduler Virtual Device

TAR Tape Archive

TTVM Time-Traveling Virtual Machine

UML Unified Modeling Language

UML User-Mode Linux

VM Virtual Machine

VM/370 Virtual Machine Facility/370

VMAN Virtualization Management

VMM Virtual Machine Monitor

VPS Virtual Private Server

WBEM Web-Based Enterprise Management

WS-Management Web Services for Management

XML Extensible Markup Language