TAMPERE UNIVERSITY OF TECHNOLOGY

OLLI HELIN

A Study on Virtualization and Energy Efficiency Using Linux

Master of Science Thesis

Examiners: Prof. Tommi Mikkonen
Ph.D. Tapio Niemi
Examiners and topic approved in the
Faculty of Science and Environmental
Engineering Council meeting on
8 February 2012

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Degree Programme in Science and Engineering
HELIN, OLLI: A Study on Virtualization and Energy Efficiency Using Linux
Master's thesis, 56 pages, 3 appendix pages
March 2012
Major subject: Software Engineering
Examiners: Prof. Tommi Mikkonen and Ph.D. Tapio Niemi
Keywords: virtualization, energy efficiency, Linux, KVM, Xen, green computing

Virtualization has in recent years risen in popularity to the extent of changing the way information technology infrastructure in enterprise data centers is built. Once known as a technique to achieve time sharing between processes, virtualization now offers flexibility in resource usage and software deployment, security, and energy savings by consolidation of many virtualized servers into a single physical one.

However, in its modern form, virtualization is still a relatively young technology. There are many studies regarding the performance of different virtualization technologies, but only a few emphasize energy efficiency. When information technology service providers invest in more server hardware, their energy expenses also rise. As optimization for energy efficiency becomes more and more important, possible power consumption overhead caused by virtualization will be an important factor when setting up virtualized servers.

In this thesis we studied virtualization using Linux with focus on energy efficiency. We conducted sets of performance tests while measuring power consumption, and assessed how virtualization affects energy efficiency. The tests included synthetic tests and more practical web server tests, with single and multiple virtual machines. We tested various configurations to find out what one should generally note when building a virtualized environment with focus on energy efficiency. All of this was done using various virtualization technologies to find out their differences regarding energy efficiency. The tested technologies were KVM, Xen, and vSphere Hypervisor.

With respect to energy efficiency or performance, we observed differences in virtualization technologies, and the same technology was not always the best in every situation. We found KVM to offer good energy efficiency, and Xen to have some trouble with recent Linux versions. In web server tests, the use of paravirtualization had almost no effect on power consumption. Processor performance states affected performance and energy efficiency. Power consumption had a tendency to be generally high with bare-metal virtual machine monitors Xen and vSphere Hypervisor. More research with a wider selection of test hardware and software is required to better define the setups and situations where this power consumption trend and the possible effect of paravirtualization on energy efficiency are observable.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Teknis-luonnontieteellinen koulutusohjelma
HELIN, OLLI: Tutkimus virtualisoinnista ja energiatehokkuudesta Linuxilla
Diplomityö, 56 sivua, 3 liitesivua
Maaliskuu 2012
Pääaine: ohjelmistotuotanto
Tarkastajat: professori Tommi Mikkonen ja FT Tapio Niemi
Avainsanat: virtualisointi, energiatehokkuus, Linux, KVM, Xen, vihreä laskenta

Virtualisointi on viime vuosina kasvattanut suosiotaan. Tämä on jopa muuttanut suurten tietokonekeskusten infrastruktuurin toteuttamistapoja. Virtualisointi tunnettiin aikanaan tekniikkana jakaa suoritinaikaa prosesseille. Nykyään se tarjoaa joustavuutta resurssien käytössä ja ohjelmistojen levityksessä, turvallisuutta, sekä energiansäästöä koontamalla useita virtuaalisia palvelimia yhteen fyysiseen.

Nykymuodossaan virtualisointi on kuitenkin suhteellisen uusi teknologia. Tutkimuksia eri virtualisointiteknologioiden suorituskyvystä on olemassa runsaasti, mutta vain harvassa on tutkittu energiatehokkuutta. Tietoteknisten palveluntarjoajien hankkiessa lisää palvelinlaitteistoa, myös palveluntarjoajien energiakustannukset nousevat. Energiatehokkuusoptimoinnin noustessa yhä tärkeämmäksi, mahdollinen virtualisoinnin aiheuttama tehonkulutuslisä tulee olemaan tärkeä tekijä virtualisoituja palvelimia pystyttäessä.

Tässä diplomityössä tutkimme virtualisointia Linuxia käyttäen painottaen energiatehokkuutta. Toteutimme suorituskykytestejä tehonkulutusta mitaten ja arvioimme virtualisoinnin vaikutusta energiatehokkuuteen. Testit sisälsivät sekä keinotekoisia testejä että käytännönläheisiä verkkopalvelintestejä, yhdellä ja useammalla virtuaalikoneella. Kokeilimme erilaisia asetuksia selvittääksemme mitä yleisesti pitäisi huomioida rakentaessa virtualisoitua, energiatehokasta ympäristöä. Kaikki tämä tehtiin käyttäen useita virtualisointiteknologioita selvittääksemme niiden energiatehokkuuserot. Testatut teknologiat olivat KVM, Xen ja vSphere Hypervisor.

Energiatehokkuuden ja suorituskyvyn suhteen havaitsimme, että virtualisointiteknologioiden välillä on eroja ja sama teknologia ei aina ollut paras kaikissa tilanteissa. KVM tarjosi hyvän energiatehokkuuden, mutta Xenillä oli erinäisiä ongelmia uusilla Linuxin versioilla. Verkkopalvelintesteissä paravirtualisoinnilla ei ollut juuri vaikutusta tehonkulutukseen ja Turbo Boost -teknologia heikensi energiatehokkuutta. Tehonkulutus oli yleisesti paljon korkeampi suoraan laitteiston päällä toimivilla virtualisointiohjelmistoilla Xenillä ja vSphere Hypervisorilla. Jatkotutkimusta tulisi tehdä laajemmalla testilaitteisto- ja ohjelmistovalikoimalla, jotta selvitettäisiin tarkemmin asetukset ja tilanteet, joissa tämän tapainen tehonkulutus ja paravirtualisoinnin mahdollinen vaikutus energiatehokkuuteen on havaittavissa.

PREFACE

This thesis was written in late 2011–early 2012 while working as a research assistant in Helsinki Institute of Physics Technology Programme GreenIT in their office at CERN. The thesis was mainly written in three places: an exciting cattle shed in Sergy, the modern Finnish office at CERN, and the ancient forests of Kangasala.

I would like to thank Jukka Kommeri, my supervisor at CERN, for his constructive comments and sharing of his knowledge concerning the local ways of destroying spare time to maximize the restoration of energy reserves for writing. I would also like to thank examiners Tapio Niemi and Tommi Mikkonen for their feedback on the way.

Special thanks to Todd Muirhead for VMware approval review. For the support, empathy, and inspiration, I would also like to thank all the people at the CERN office, all my friends, Klipsch, and JC Denton.

February 18th 2012
olli.helin@iki.fi

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

4.2BSD	Berkeley Software Distribution 4.2, an operating system.
ACPI	Advanced Configuration and Power Interface, a specification for device power management.
AMD	Advanced Micro Devices, Inc. A semiconductor company.
BIOS	Basic Input/Output System. The software code which initializes system devices when a computer powers on.
CERN	European Organization for Nuclear Research, an organization operating the world's largest particle physics laboratory.
CPU	Central Processing Unit. The CPU carries out the instructions of computer programs.
DMA	Direct Memory Access, a feature that allows system memory access for hardware devices independently of the CPU.
ext4	Fourth extended filesystem, a journaling file system for Linux.
Gb/s	Gigabits per second.
HDD	Hard Disk Drive.
HTTP	Hypertext Transfer Protocol.
IA-32	Intel Architecture, 32-bit. A complex instruction set computer architecture.
IDE	Integrated Drive Electronics, a standard for connecting disk drives.
IOMMU	Input/Output Memory Management Unit. Connects DMA-capable devices to the main memory.
KVM	Kernel-based Virtual Machine.
LTS	Long Term Support, a software release with longer support length than normally.
LXC	Linux Containers.
MFLOPS	Millions of floating-point operations per second.
ms	Millisecond.
$\bar{P}$	Mean power consumption.
POPF	Pop Flags, an x86 instruction.
RAM	Random Access Memory. In this thesis, refers to the computer's main memory.
SCSI	Small Computer System Interface. A set of standards for computer device connecting.
SLC	Scientific Linux CERN. A Linux distribution in use at CERN.

Continued on next page

List of symbols and abbreviations—continued from previous page

σ	Standard deviation.
SPEC	Standard Performance Evaluation Corporation, an organization that produces performance benchmarks for computers.
TCP	Transmission Control Protocol.
USB	Universal Serial Bus.
VCPU	Virtual Central Processing Unit.
VMFS	Virtual Machine File System, a file system developed by VMware, Inc.
VMM	Virtual Machine Monitor.
W	Watt, a unit of power.
$\bar{x}$	Arithmetic mean of samples.

1. INTRODUCTION

In recent years, virtualization has been changing the way information technology infrastructure in enterprise data centers is built. The need for large data centers arose due to demand for computational power [1]. This computational power goes to running services. Cloud infrastructures like Amazon Elastic Compute Cloud [2] and Microsoft Online Services [3] provide resources for various computing needs. Google and many others offer software as a service directly to a user's web browser [4]. Shopping has been shifting more and more online [5].

All these services require large amounts of servers and flexibility to satisfy the demands of an ever-growing userbase. This growth has had a side effect. Lately, energy expenses in the data centers have been rising to the extent of possibly surpassing the actual hardware costs [6]. All this calls for more energy efficient computing.

In many cases of real life data servers, however, energy efficiency measures are conducted only when the infrastructure has already reached its maximum capacity [7]. Even then, the focus of optimization has mostly been in hardware and infrastructure, not in operational methods, operating systems, or software [8]. Harizopoulos et al. [6] note that hardware optimization is only part of the solution and software also must be taken into account when pursuing energy efficiency. Barroso and Hölzle [9] also argue that servers require new energy efficiency innovations in addition to the energy efficient hardware. Venkatachalam and Franz [10] surveyed power consumption reduction in circuit techniques and hardware features among others, and also recognized the importance of software in energy efficiency.

Virtualization is one solution to the problems. Using virtualization enables one to cut costs and enhance energy efficiency. There have been a lot of synthetic performance tests using virtualized systems. For example, Padala et al. [11] studied performance degradation of applications on virtualized systems against a base Linux system using Xen and OpenVZ. Tafa et al. [12] studied the performance of the same virtualization technologies during live migration. Soltesz et al. [13] studied the performance advantages of container based virtualization over hypervisors. There are also lots of studies on improving performance in specific situations. For example Wang et al. [14] developed a faster way for inter-virtual machine communication using Xen. Hu et al. [15] introduced a scheduling model to improve input/output performance of virtualized systems.

However, as virtualization in its modern form is still a relatively young technology, there are not that many studies between different virtualization solutions with an emphasis on energy efficiency. Bearing in mind the generally recognized need for software that is optimized for energy efficiency, possible power consumption overhead caused by virtualization will be an important decision factor when setting up virtualized servers.

This thesis is a study on virtualization with focus on energy efficiency. Measuring energy efficiency for a server requires knowledge of performance and power consumption characteristics. We conducted sets of tests while measuring power consumption. From the test results we assessed how virtualization affects performance and consequently energy efficiency. We tested various configurations to find out what one should generally keep in mind when building a virtualized environment with focus on energy efficiency. All of this was done using various virtualization technologies to find out their differences regarding energy efficiency.

The structure of the thesis is as follows. We first take a look at the history of virtualization in Chapter 2. We also discuss motivation for virtualization and why energy efficiency is important. We then introduce the specific virtualization technologies studied in this thesis. In Chapter 3 we describe the test environment and testing methodology. Results from the tests are presented and discussed in Chapter 4. Finally in Chapter 5 we summarize the results and discuss further research on energy efficiency.

2. BACKGROUND

In this chapter we will discuss the concept of virtualization. What virtualization is and what is its history is discussed in Section 2.1. Motivation for virtualization is discussed in detail in Section 2.2; server consolidation, an energy efficiency related reason to virtualize, is further discussed in Section 2.3. In Section 2.4 we introduce virtualization technologies currently in use and relevant for this thesis. In Section 2.5 we will discuss cloud computing, a strongly commercial motivator and area of use for virtualization. In Section 2.6 the hardware features related to energy efficient computing and relevant for this thesis are introduced. Finally, in Section 2.7 we introduce the virtualization solutions which are studied in this thesis.

2.1 Virtualization

As a generic term, virtualization means creating a virtual version of something, a simulation of the real version [16]. In computing, it is "a technique for hiding the physical characteristics of computing resources from the way in which other systems, applications, or end users interact with those resources." [17]

Virtualization technologies are widely used in a variety of areas such as multiple operating system support, server consolidation, transferring systems from one computer to another, secure computing platforms and operating system development [18]. One of the carrying ideas in virtualization is to permit multiple operating systems to execute operations directly on hardware, yet leave ultimate control to the virtualization implementation, so that sensitive operations can not affect other guest operating systems [7]. An example of these sensitive operations are direct memory access (DMA) requests for input/output. This arbitrating of accesses to the physical resources is done in a new layer of software between the hardware and operating systems [19]. Virtualization may thus be characterized also by the addition of this layer.

Virtualization technology emerged in the late 1960s to improve hardware utilization. Computers back then were very expensive and not so readily available: computing was done with large mainframe hardware [20]. There existed a problem with time sharing: as operating systems were not multitasking and software were written so that they expected to have all the resources of a computer, there was a need for something to make the single computer appear as many. Virtualization was the answer for that: for example on a single IBM System/360 one was able to run fully isolated environments in parallel so that each environment thought they actually had the whole mainframe at their disposal [21]. This time sharing was done by so called Virtual Machine Monitors (VMMs), also known as *hypervisors*.

As hardware costs fell and multitasking operating systems became commonplace in the 1980s and 1990s, the need for the type of time sharing used with the IBM System/360 disappeared—in actuality, computer architectures no longer provided the appropriate hardware to implement the VMM technology efficiently [20]. History seems to repeat itself, as even with modern hardware virtualization has been the answer to better utilize it. Nowadays time sharing is not the main motivation for virtualization, however. That part is already covered by multitasking operating systems. Virtualization brought with it other benefits which hold true even today.

Nowadays a hypervisor is commonly a solution for security and reliability [20, 22]. This is a direct consequence of isolation: virtualizing two machines isolates them; they behave as two. So if one was to have a maintenance break or proper function was ceased due to faulty software, the other would continue functioning. Indeed, security in virtualization has become a popular topic in recent years not only because of the security virtualization can offer, but also because of the security threats a hypervisor itself must face [23].

Basically every component of a computer can be virtualized and when virtualizing a computer one actually virtualizes many things—after all, a computer is merely a word for programmable machine. This machine consists of things such as

- central processing unit (CPU), which carries out the computations
- memory, also known as random-access memory or RAM for short, the working space for applications and the operating system
- input/output (I/O), or the means by which a computer is able to exchange information in and out of the computer, for example by storing data on a hard disk drive
- network adapter, which allows multiple computers to form a network for sharing of resources and information.

Even though one is generally able to virtualize everything, it is not always purposeful to do so. The question is not what one is able to, but what is beneficial to or harmful to virtualize. Gammage & Dawson argue that virtualization is the primary cause of I/O performance degradation; also where an application needs confirmation that hardware operations have indeed been completed in specific time, it might be impossible to utilize virtualization in such cases [24].

2.2 Reasons to Go Virtual

As discussed in the previous section, the reasons which led to virtualization are now mostly outdated and the current reasons to virtualize are far more complex. For example, virtualizing certain components of a computer one achieves certain effects, some of which may benefit one situation and others which may benefit another situation.

A CPU architecture is virtualizable if it is possible to execute a virtual machine directly on the real machine, while still letting the hypervisor to retain control of the CPU [20]. Virtualizing the CPU allows multiple operating systems to share processor resources efficiently. This satisfies the requirements of such infrastructures where for example a mail server requires Windows Server and a database runs on Solaris [22]. Also, using emulation one is able to run software written for different computer architectures [25]. This may be beneficial when testing multi-platform software or when using software for which there no longer exists the required hardware. Virtual CPUs are also utilized in high performance computing in cases where processing jobs do not take full advantage of multicore processor architectures: one may deploy for example a virtual machine per core to better share resources among jobs [26].

Virtualizing memory allows the memory allocated for virtual machines to exceed the memory size of the actual hardware [20]. This in turn enables for example running two operating systems each requiring three gigabytes of memory on a hardware which only has four gigabytes instead of the combined amount of six gigabytes.

Virtualizing the I/O allows the system data storage model to be decoupled from its physical realization. For example, the virtual machine might look like it has a normal hard-disk drive with a couple of partitions, but in reality it could be using just an abstraction of a storage device, behind which exists redundant storage situated in completely different geographical location. This results in flexibility and robustness in forms of runtime expansion and transferring and replicating the system to another computer. An example of I/O virtualization is illustrated in Figure 2.1. [27]

Virtualizing the network allows one to create a network with separate machines within single hardware. These machines may use inter-virtual machine communication mechanisms which for the user are completely transparent and seem like a normal network device, but with huge performance benefits between the virtual machines compared to using real network adapters. [14]

Basically, when something has been virtualized it is no longer bound to the physical realization. This in turn enables N:N relationships between applications and and hardware: one is able to run multiple isolated applications on a single shared resource, but also a single application on multiple physical resources [22]. One of these uses of multiple physical resources is live migration, which nowadays is one

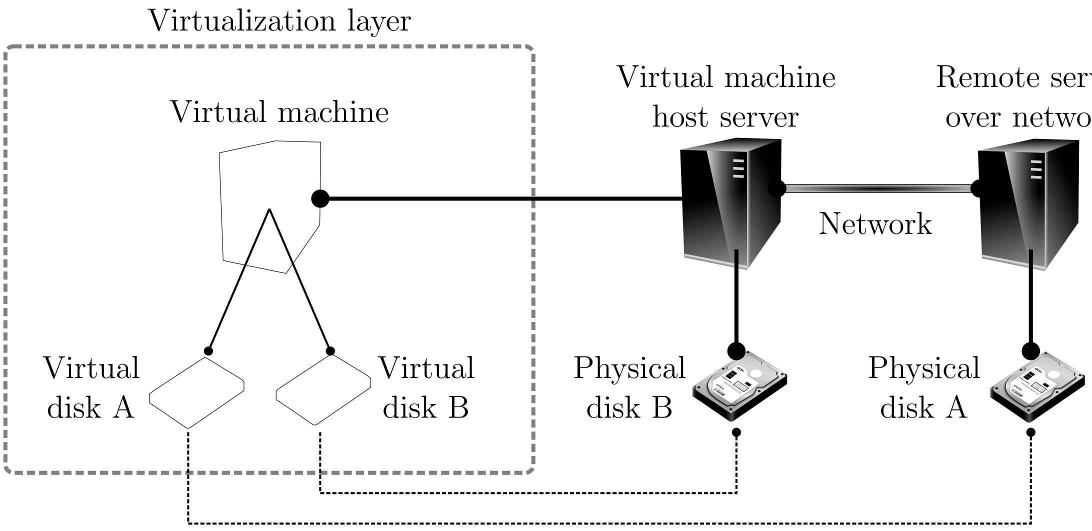

Figure 2.1: Virtualization decouples the system model from the physical realization. In this case the disks have been virtualized: the virtual machine sees and writes both virtual disks A and B in the same way, but the physical disk A is actually located elsewhere physically and accessed over network connection.

of the most important uses of virtualization techniques [12]. In live migration one transfers a virtual machine from one physical machine to another without interrupting application execution, enabling for example the re-purposing of server capacity to better meet the needs of application workload owners [28].

Another use of virtualization is in application deployment and management. Virtualization offers uniform application deployment environment independent of any hardware, avoiding problems software engineers might have should they have to deal with different hardware configurations. This use of virtual machines for uniform application deployment can also be utilized in disaster recovery: if something is broken, one just has to drop in the model virtual machine image and restart the machine. Similar approach may be utilized in testing phase: when testing different software configurations, one starts the corresponding virtual machine and when required, going back to the starting point is trivial. [22]

Virtualization also offers security. Usually operating systems provide relatively weak forms of isolation between applications; for example, the file system and process identifiers are normally shared. When using virtual machines, one of the goals of hypervisor is to provide full isolation between the machines. Therefore, by running one application in one virtual machine and some other application in another virtual machine, the applications cannot affect each others functionality. The virtual machines may also have their own networks, which obviously has a positive effect on security should one virtual machine become compromised: the other's network still remains uncompromised. [13]

Finally and most interestingly as far as this thesis is concerned, virtualization enables huge energy savings by means of server consolidation, which in the past years has been one of the biggest uses of virtualization [22]. Server consolidation is further discussed in the following section.

To sum up, the main reasons for modern use of virtualization are energy efficiency by server consolidation, elasticity in resource usage, security, and easier deployment of software, with current trends leaning more towards consolidation and future trends into other areas.

2.3 Server Consolidation

Server consolidation by virtualization means running multiple applications in separate virtual containers hosted on single hardware. These virtual containers can be full virtual machines and the applications can be operating systems: in essence, one is able to consolidate multiple virtual machines into one real machine. In enterprise data centers, this has become an integral part of information technology planning to more efficiently utilize hardware and to reduce costs [11, 28].

When it comes to server hardware costs, it was noted in a study conducted within an European Union programme Intelligent Energy Europe that many businesses do not consider energy costs within total cost of ownership [7]. The same study suggests that energy saving potentials of sixty percent could be achieved by applying and optimally managing efficient information technology hardware and infrastructure. One factor contributing to the huge energy savings is the fact that in data centers, cooling and other infrastructure actually uses fifty to hundred percent as much energy as the servers themselves [7, 29]. This is also one of the reasons why for example in a computer centre at CERN there are half empty computer racks as seen in Figure 2.2; lack of sufficient cooling power limits the amount of computers. Consequently and not surprisingly, it has also been suggested that energy expenses could actually be the dominant factor in the total cost of ownership [30].

When studying server utilization figures, we notice that the lowest energy efficiency region is also a very common one under normal server operation [9]. Padala et al. suggest a typical average server utilization of below thirty percent [11]. Vogels from Amazon.com mentions utilization levels as low as five percent, the typical being just below twenty percent. Barroso and Hölzle from Google note that in a highly tuned environment such as Google's, a server might have a typical utilization level between ten and fifty percent [9]. To link these utilization levels to system energy efficiency figures, we take a look at two typical results from Standard Performance Evaluation Corporation's (SPEC) server side Java benchmark. SPEC is a consortium producing computing performance benchmarks. Their SPECpower_ssj2008 benchmark evaluates the power and performance characteristics of server computers

Networking, then and now
La grille, c'est quoi?
What is the Grid?
Le web et la grille - quelle est la différence?
The Web and the Grid - what's the difference?
Le stockage, jadis et aujourd'hui
Storage, then and now

Figure 2.2: CERN computer centre. Some of the server racks are mostly empty because there is not enough cooling power to accommodate more computers.

by using server side Java workload [31, 32]. The results are given for various system load levels as server side Java operations per watt of energy consumed, SSJ_ops/W.

In Figure 2.3 we see the performance to power ratio and average power consumption figures of a modern server; the data is from SPECpower_ssj2008 results database [33]. The server in question is IBM System x3200 M2. It is an Intel Xeon based dual-core server running Microsoft Windows Server 2003. We notice that the best performance to power ratio is achieved with maximum system load. In the aforementioned operating region of thirty percent utilization, the performance to power ratio is already less than half of the maximum: energy efficiency has dropped dramatically. Even when idle, the server uses two thirds of its maximum power. From Figure 2.3 it is obvious that with typical utilization levels, the server would be running very inefficiently.

In Figure 2.4 are another SPECpower_ssj2008 test results [34]. The server in question is Fujitsu PRIMERGY TX300 S6 with two hexa-core Intel Xeon processors. Compared to the IBM above, the PRIMERGY is a server for more heavy loads. The performance to power ratio is more logarithmic compared to the linear shape of IBM, resulting in better energy efficiency in the fifty percent system utilization region. When falling down below twenty percent utilization, however, energy efficiency degrades again.

Now if one was to consolidate for example three servers with an average utilization

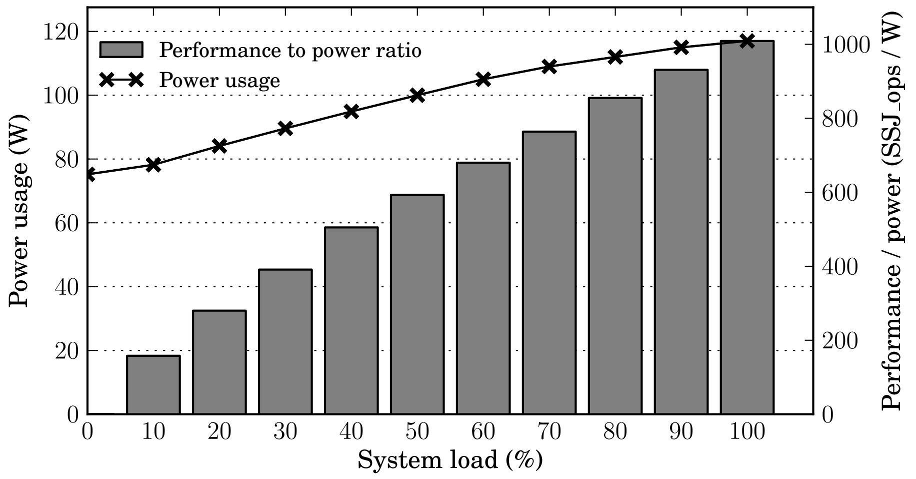

Figure 2.3: Performance to power ratio and average power consumption figures of IBM System x3200 M2 server. Maximum efficiency is achieved at a hundred percent system load. Efficiency drops approximately in a linear fashion as system load gets lighter. Figure data courtesy of SPEC [33].

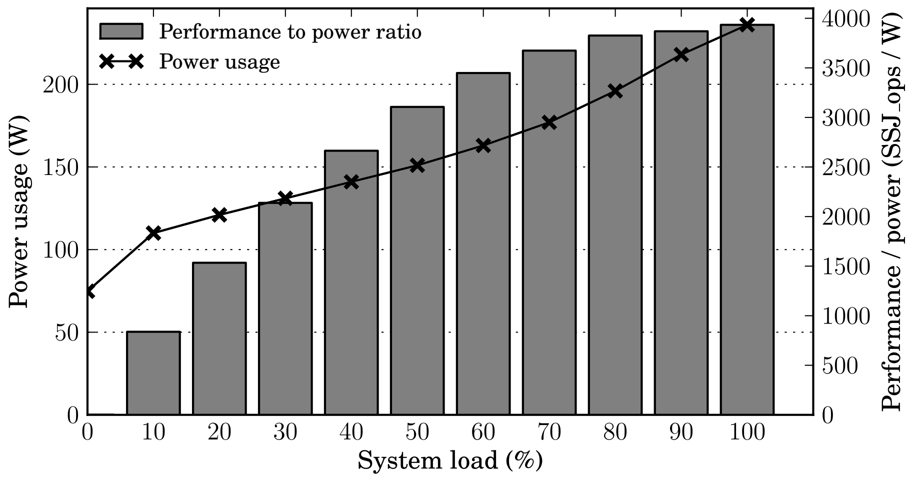

Figure 2.4: Performance to power ratio and average power consumption figures of Fujitsu PRIMERGY TX300 S6 server. Maximum efficiency is achieved at a hundred percent system load, but the performance to power ratio stays relatively high even with system loads as low as forty percent. Figure data courtesy of SPEC [34].

of twenty percent into one single hardware, one would get a server with an average utilization of sixty percent. This is of course a naive calculation not taking into account the different kind of load levels and virtualization overheads, but the idea is sound: by server consolidation, one gets to the utilization region where energy efficiency is better.

Full hundred percent average utilization is never the goal, however. Server workloads fluctuate over time and utilization spikes do occur [22]. A web server with high average utilization might have trouble meeting its service-level agreements in throughput and latency, for example [9]. The lack of leeway would also make maintenance tasks difficult. For environments running various applications, Vogels considers an average utilization between fourty to fifty percent to be excellent results [22]. Finally, despite all these advantages in energy efficiency offered by server consolidation, it is completely normal to run just one virtual machine on a physical server. The focus is then not in energy efficiency, but scaling potential and speeding up deployment of applications [22].

2.4 Virtualization Technologies

A virtual machine monitor is a software layer which separates underlying hardware from the software running on top of it, creating an abstraction of the hardware for a virtual machine [35]. The abstraction may look similar independent of hardware, transforming the view of hardware as a set of specific components into a view of hardware as a pool of resources [20]. The virtual machine monitor may then map these resources to the virtual machines running on top of it as requested, providing complete encapsulation of a virtual machine's state. Thus it is possible to change the underlying hardware and continue the virtual machine's operation normally. In practice, this could mean for example migrating the virtual machine from one server to another. Virtual machine monitors are sometimes also referred to as hypervisors.

Traditionally, virtual machine monitors have been split into two groups: Type I and Type II [35]. Those of Type I run directly on the host's hardware and are known as bare-metal hypervisors for that. Figure 2.5 shows an example situation where this type of virtual machine monitor separates two operating systems from hardware. Examples of these kind of virtual machine monitors include VMware ESXi [36] and Microsoft Hyper-V [37]. Type II virtual machine monitors on the other hand run within an operating system. They are known as hosted hypervisors. Examples of these are the Oracle Corporation's VirtualBox [38] and VMware Workstation [39].

Two of the central software in this thesis are the Kernel-based Virtual Machine (KVM) [40] and the Xen hypervisor [41]. They are both quite hard to categorize either as Type I or Type II. KVM turns a Linux-kernel into a Type I hypervisor, even though one could argue that KVM runs on a Linux distribution, making it Type

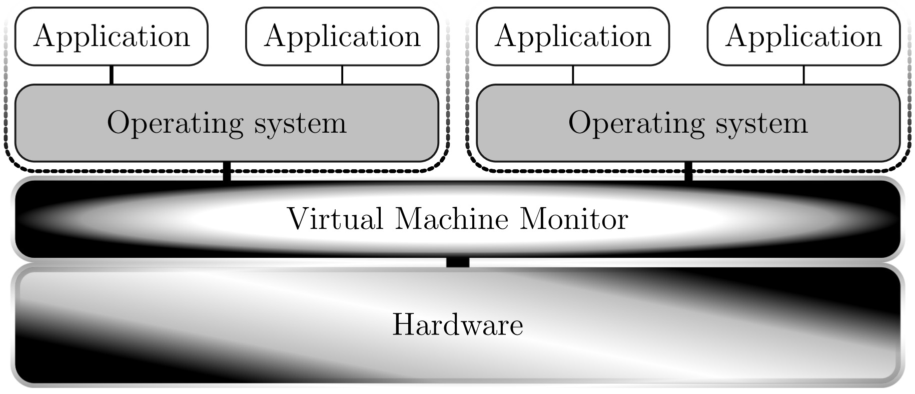

Figure 2.5: The virtual machine monitor decouples underlying hardware from the operating systems running on top of it. In an operating system's point of view, it has the computer's hardware resources completely under its control, while in reality the resources are shared between the operating systems.

II. The Xen hypervisor, on the other hand, is of Type I but then again requires a privileged operating system to handle the virtual machines, making it in a sense a Type II hypervisor. This is why the labeling whether a hypervisor is of Type I or II does not really mean anything per se and is mainly used to describe the general nature of the hypervisor. A bare-metal hypervisor might sound as if it had a smaller privileged codebase as it does not have the burden of underlying operating system, so it might be more secure and perform better. A hosted hypervisor on the other hand might be easier to set up on a computer currently in production use, as it might not require any modifications to the underlying operating system and device drivers for the hypervisor would be readily available.

The addition of this software layer is likely to create some problems, however. The problems arise from the code a central processing unit is executing. This code is categorized into different privilege levels. For example, the IA-32 instruction set architecture, which is the most common in the world [42], has four privilege levels, ranging from most privileged to least privileged [43]. For the sake of simplicity, let us consider there are only two privilege levels: privileged and unprivileged. Usually in the privileged level lies the operating system kernel code and device drivers, all the code which needs direct access to hardware. In the unprivileged level lies normal user applications, which use the services provided by operating system to access the hardware: for example, when saving a text document from an editor, the editor is not directly communicating with the hardware, but uses the system calls provided by the operating system to do so.

Running a virtual machine requires the guest operating system's kernel to be run in unprivileged mode, as the virtual machine monitor is running in privileged mode. In IA-32 there are instructions which execute differently depending on whether they are run in privileged or unprivileged mode: for example, when run in unprivileged mode, the pop flags (POPF) instruction does not clear a certain bit in the processor's status register as it normally should when run in privileged mode. If one was executing this instruction in a virtual machine, the result would be erroneus as the processor's status would not be what it should. There are also other challenges related to privileges certain instructions have access to. [20]

To overcome these challenges, different techniques have been developed. One possibility is binary translation: the virtual machine monitor translates all instructions from the guest operating system so that the end result is correct. These kind of software methods are usually slow, however. A widely used solution offering much better performance is paravirtualization. In paravirtualization, the guest operating system kernel is modified to be aware of the fact it is running on a virtual machine [12]. With that knowledge and co-operation with the hypervisor, it is possible for the guest operating system to execute with near native speed without the instructions which are hard to virtualize [44]. This technique suffers from the fact that the guest operating system can not be run without modifications, however. To fully virtualize IA-32 or other architectures from the x86 instruction set architecture family and to maintain compatibility with unmodified operating systems, processor vendors developed x86 hardware virtualization extensions. To name two, AMD's solution is known as AMD-V and Intel's VT-x, respectively. The main thing they add is two operating modes for the central processing unit, each mode with their own privilege levels, so that both the virtual machine monitor and the guest operating system are able to run with their usual privilege levels but with different operating modes [19]. This enables normal unmodified guest operating system functionality while letting the virtual machine monitor retain ultimate control.

Albeit using the hardware virtualization extensions is the preferred way to run unmodified guest operating systems, paravirtualization still has a lot of uses as the operating system kernel is not the only thing that can be paravirtualized. One must bear in mind that also the device drivers are part of the privileged code. By using paravirtualized device drivers for network and disk access for example, one is able to gain the performance benefits compared to full virtualization where the instructions by device drivers must go through an extra software layer. Using paravirtualized device drivers is separate of paravirtualizing the whole operating system: for example, even if running Windows XP guests on the Xen hypervisor requires full virtualization, there exists paravirtualized Xen network and I/O drivers for Windows XP to bridge the performance gap [45]. As with processor virtualization, hardware techniques to speed up the operation of fully virtualized devices have been developed, for example AMD-Vi and Intel VT-d [46].

Sometimes it might be beneficial not to virtualize the hardware but the operating system running on it. This is called operating system level virtualization or operating system virtualization. The idea is not to waste resources by creating multiple virtual machines with the same operating system in them, but to create isolated virtual environments within one common operating system, thus saving the resources needed to run multiple operating system kernels. This is especially true when it comes to memory usage, as it might be hard for the virtual machine host to know how the guests are using their memory and redundand copies of identical code and data between the guests might be stored in the host's memory [20]. In general, when comparing hypervisor based virtualization and operating system virtualization, the former brings along it a higher performance overhead, but might provide better isolation between the virtualized environments [11].

The virtualized environments which are created using operating system-level virtualization are usually called containers or jails. Examples of implementations include chroot, LXC, Linux-VServer and OpenVZ. Although most container based virtualization technologies are relatively new, chroot is an exception dating all the way to the early 1980s when it was introduced in 4.2BSD operating system [47].

In some cases, it is possible to achieve near hypervisor level isolation with operating system virtualization but with performance benefits. Also, whether one wants to give weight on performance or isolation depends on the case. For example, if a server is running applications on behalf of two independent organizations, it is critical that the applications minimize information sharing, making hypervisor a suitable choice. On the other hand, if strict isolation is not needed or if the applications do need to share data, it is possible to achieve noticiable performance advantages with container-based operating system virtualization. [13]

Another technique which is usually done using just one operating system is sandboxing. In sandboxing, the idea is not so much to virtualize something as to provide means of isolation for security purposes. As the name suggests, the goal is to provide a sandbox, a place where an application can be executed while isolated from the rest of the system, just like a child can safely play in a sandbox.

As the term sandboxing basically means just isolating an application and does not specify any technique per se, virtualizing a full computer system just for running one application could be called sandboxing. This is rarely the case, however, as that would mean wasting resources. Usually sandboxing is achieved with very little overhead [48].

For some security critical situations, even a fully virtualized computer system might not be enough. For example, a malicious application running on a virtual machine might still have free access to the network just as if run directly on hardware. The obvious solution is to combine both virtualization and software sandboxing to reach maximal isolation [49]. This kind of combination is used for example in cloud computing in isolating development environments for different users [50]. Sandboxing in general is widely used in software development for testing purposes. For example, PayPal provides a sandbox in which developers may test their applications related to money transfers, which on a live website would be problematic to test [51].

2.5 Cloud Computing

Cloud computing or simply a cloud is the realization of the paradigm of shifting the location of computing infrastructure to the network, with the aim of reducing both software and hardware resource management costs [52]. Usage of clouds can be split into three scenarios: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) or Software as a Service (SaaS) [53]. In IaaS scenario, virtualization is used to split, resize and assign hardware resources dynamically into virtualized systems as the customer wishes. A simplified example of this scenario is illustrated in Figure 2.6. At this point it is important to note that the foundation of cloud computing is formed by virtualization, as it is the technology which provides the capability to handle the resources as stated above [54]. Virtualization is also the primary security mechanism in today's clouds [55].

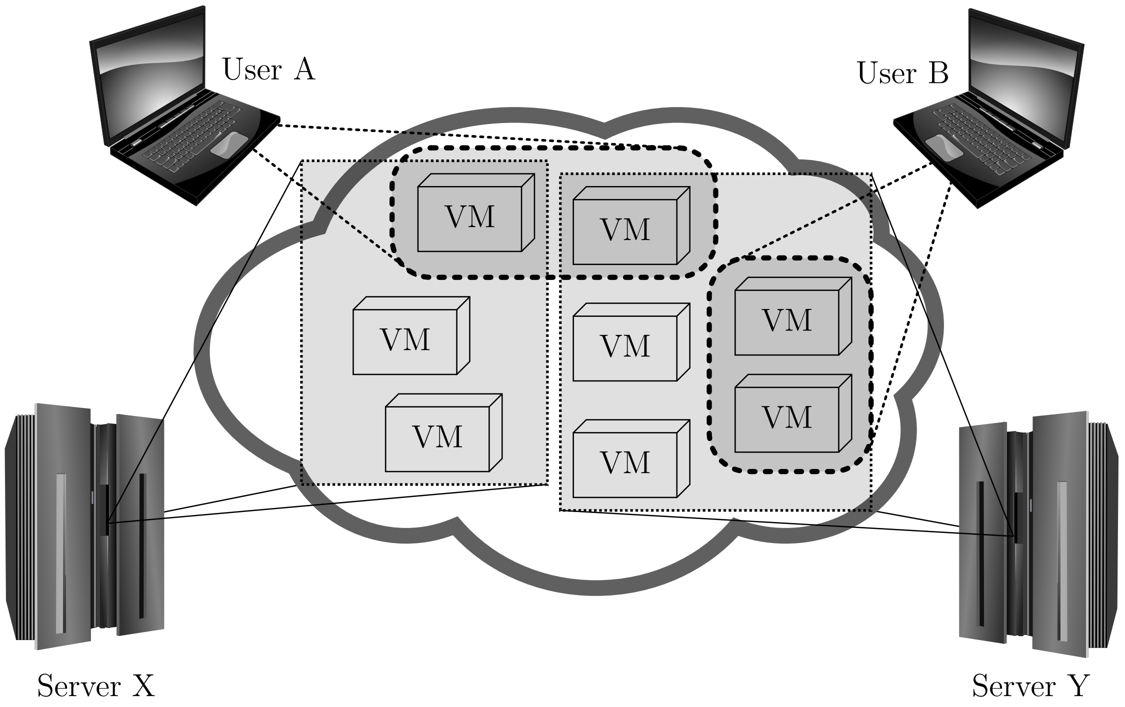

Figure 2.6: Cloud computing as we know it would not be possible without virtualization. Servers X and Y host multiple virtual machines inside the cloud. Clients A and B are using the virtual machines without any knowledge of the underlying hardware infrastructure.

In PaaS scenario, instead of offering virtual hardware, service providers offer a software platform while the managing of required hardware resources is abstracted behind the platform. An example of PaaS is the Google App Engine which developers may use for building public or in-house web applications with high scalability and availability [56]. In this sense, cloud computing leverages virtualization at multiple levels as not only the hardware is virtualized but also the application platform [54].

In SaaS scenario an application, for example a word processor, is run in the cloud and rendered to the end user's screen. This usually happens inside a web browser window [52]. SaaS is an alternative to locally running an application, but possibly with added abilities: for example using Google Docs, it is possible for multiple users to edit a document simultaneously over the web and see the changes in real time [57].

For end users, the cloud details are usually well abstracted behind easy-to-use user interfaces. For example the Amazon Elastic Compute Cloud has a web service interface which allows the user to set up and configure computing resources as required [58]. Clouds are a big driver behind the spread of virtualization techniques, as cloud computing is getting more and more advanced and popular [52, 54, 59].

2.6 Hardware Features

Threads are one approach to concurrent programming: they are sequential processes that share memory [60]. If an application is programmed using threads, it obviously executes faster if there are more than one processor running the threads. But more processors means that more electric power and hardware is required. If an application is single-threaded, all the other processors would be just a burden for energy efficiency. Even though modern simultaneous multithreading processors are able to execute both multi-threaded applications and parallel single-threaded applications efficiently [61], there comes times when on one hand it would be beneficial to be able to shut down or otherwise set to an energy saving state the idle parts of a processor, and on the other hand somehow improve the performance of the already fully utilized parts. Special hardware features have been developed to do just that. In this section presented are the features relevant to this thesis.

The faster clockrate a processing core runs at, the more power it consumes and thus the more it heats up. Generally, the power consumption per operation is considered to be directly proportional to the product of core voltage to the second power and core frequency. Obviously, the performance depends on the clock frequency. If voltage is the more defining factor, then why do not we just lower it? The way processors work, generally the higher the clock frequency, the more voltage they need. One can not have high clocks without high voltage, and one can not lower the voltage to save energy without also lowering the clock frequency and thus performance. This is why voltage and frequency is scaled more or less at the same time, as lowering just the frequency would still leave high power consumption because of high voltage. [62]

While a processor has just a little load on its cores, it is thus energy efficient to scale down the frequency and voltage of the cores. Intel calls their implementation of this technique Intel SpeedStep [63]. AMD's implementation with desktop processors is called Cool'n'Quiet; AMD claims over 55 percent power savings are achievable with all their power saving technologies enabled [64]. In this thesis, some of the tests were conducted using the Intel SpeedStep technique for comparison to without using it. Another technique to save energy with lightly utilized processors is processor power state switching. The Advanced Configuration and Power Interface (ACPI) specification defines power states for processors supporting the specification, so that operating systems can set processors to more energy saving states when the processors are not needed [65]. Effectively, parts of a processor are turned off completely.

When running applications which utilize only one or just a few cores of a multicore processor, it would be beneficial to improve the performance of those cores. As processor core frequencies can be scaled down for power saving purposes, they can also be scaled up for performance boost. Intel's implementation of this is called Intel Turbo Boost [66]. It works so that in a multi-core processor, if there are cores with low load, some cores can be turbo boosted. If all cores are under heavy load already, none of the cores can be turbo boosted as it would mean the processor's thermal limits would be exceeded.

Turbo boost was extensively used during the energy efficiency tests in this thesis. The energy efficiency potential behind using turbo techniques lies in the fact that even though the power consumption of a processor is higher with higher clock frequencies, performing the operations takes less time. Energy consumption is dependant on power consumption and used time, so higher energy efficiency is possible even with higher power consumption if significantly less time is used for the task.

Core frequency is not the only thing which can be optimized. What happens during a clock cycle is another important factor. One approach in optimizing a core's functionality is hyper-threading.

The Hyper-Threading Technology by Intel was first introduced in their Xeon processors in 2002. In hyper-threading the idea is for two logical processors to simultaneously share the execution resources of one physical. To software, these two logical processors appear just as any two processors would. The idea is to better utilize the computing resources of a processor with low complexity physical additions to the processor die, keeping power usage and material requirements low. For example, if two threads were being executed, a slow thread might get an unfair share of resources, preventing the faster thread from making rapid progress. [67]

Hyper-Threading Technology is also used in modern Intel server processors [68]. Although using hyper-threading is in some cases known to have performance near traditional symmetric multiprocessing using multiple processors but with only little impact on power consumption [69], there are also reports where using hyper-threading has had little effect on performance and thus energy efficiency [70]. In the tests conducted for this thesis, hyper-threading was disabled to eliminate one factor of unpredictability.

2.7 The Virtualization Lineup

The virtualization software and technologies which will be studied closer and tested are introduced in this section.

2.7.1 KVM

Kernel-based Virtual Machine (KVM) is a virtualization solution for Linux on x86 hardware with hardware virtualization extensions. KVM consists of a loadable Linux kernel module and another processor specific hardware virtualization extension module. There currently exists two of the latter: one for AMD processors using AMD-V and one for Intel processors using Intel VT-x. KVM uses the regular Linux scheduler and memory management: each virtual machine created is seen as a process in the host operating system, which acts as the hypervisor. Even though KVM is intended only for Linux as host, it is able to run most modern operating systems as guests. [40]

To actually create virtual machines using KVM, one also needs a user space application for this: QEMU. QEMU is a generic, open source machine emulator and virtualizer [25]. As a machine emulator, QEMU emulates a whole computer including various processors and devices, allowing it to run unmodified guest operating systems. As KVM allows a user space program to access the hardware virtualization capabilities of the processor, KVM enables QEMU to skip emulating the processor and use hardware directly instead.

To further improve speed it is also possible to use paravirtualized disk and network drivers in the guest operating system. QEMU with KVM uses VirtIO to achieve this. VirtIO is a Linux standard for network and disk device drivers capable of cooperating with the hypervisor [71]. As is the case with KVM, VirtIO drivers are readily available in the Linux kernel.

2.7.2 Xen

Xen is an external hypervisor, a layer of software running on computer hardware replacing the operating system. The Xen hypervisor is one of three components required when using Xen for virtualization: the other two are Domain 0 (Dom0, or the privileged domain), and one or more DomainUs (DomU, or an unprivileged domain). Dom0 runs on the hypervisor with direct hardware access. It is responsible for managing the unprivileged DomUs, which run the guest operating systems and have no direct access to the hardware. A system administrator manages the whole computer system by logging into Dom0. [41]

Even though Xen exists in the vanilla Linux starting from kernel version 3.0, it does not require DomUs or even Dom0 to be running Linux. DomUs' operating systems can be run unmodified when using the system's hardware virtualization extensions, or they can be run paravirtualized, in which case the guest operating system is modified to be aware of it is running on Xen hypervisor instead of base hardware. Paravirtualized Xen drivers for example for disk are included in the Linux kernel and available for Windows guests.

One of Xen's strengths is said to be its small trusted computing base. The Xen hypervisor itself is relatively small, so it is thought to be trustworthy of correct and secure operation. However, Xen requires the privileged Domain 0, which contains a full operating system with all its possible security and reliability problems. In this sense, Xen is equal in complexity to for example KVM. [72]

2.7.3 VMware ESXi

Much like Xen, the commercial VMware ESXi is a hypervisor running directly on hardware. Forming the architectural backbone of the current VMware vSphere product line, it was formerly known as ESX (without the i) and used the Linux kernel as part of its loading process. In ESXi the Linux part has been removed, and the ESXi is reliant on no specific operating system. Formerly the management was done via service console in a similar way as in Dom0 with Xen, but now all management is done with remote tools. The goal has been to reduce codebase to improve security. [36]

As ESXi uses its own kernel, it needs to have its own specific hardware drivers, too. Compared to Xen and KVM which are able to use the huge driver base of Linux, ESXi has to use its own supply of drivers. VMware claims this is on the other hand one of its strong points, not relying on generic drivers but using optimized drivers for supported hardware. ESXi enables for example quality of service based priorities for storage and network I/O. [36]

In this thesis, the freely available ESXi based VMware vSphere Hypervisor was used along with VMware vSphere Client to manage it. The vSphere Hypervisor is henceforth referred to as ESXi in this thesis.

2.7.4 Chroot

Chroot is the name for both the Linux system call and its wrapper program. Chroot is a form of operating system level virtualization, where one creates so-called chroot jails to isolate environments. The only thing chroot does is it changes the root directory of the calling process and consequently all children of the calling process. This seemingly trivial effect actually has a big impact on how one is able to run applications. Security can be enhanced by the isolation offered by chroot, and for example many network daemons can run in chrooted environment [73].

In our tests chroot is used to take advantage of an isolated Scientific Linux CERN 5.6 environment to run Apache web server with Invenio database already set up. Therefore there was no need to install web server software or to set up the database to the chroot environment's host operating system, Ubuntu. This eliminated for example the possible software library conflicts the Invenio database might have had with a different operating system than the one it is intended to be run on.

2.7.5 LXC

Linux Containers (LXC) is another operating system level virtualization mechanism. Compared to chroot, LXC extends the isolation capabilities by adding for example network namespace isolation. Network namespaces are private sets of network resources associated with certain processes: processes in one network namespace are unable to access network resources in another network namespace. This has immediate security benefits: if a server is compromised, the rest of network system will remain unaffected. Also traffic control and resource management is more flexible and more easily controllable. [74]

LXC is still an emerging virtualization technology and testing it was limited to experimenting with its setup. Unfortunately, with LXC version 0.7.5 we were unable to start the containers with the test operating system. Virtually any reports of LXC in production use are also yet to be found. The reason why LXC is considered the future of container based virtualization with Linux is that older solutions such as OpenVZ and Linux-Vserver depend on kernel patches to work. LXC uses the Control Groups mechanism found in the relatively new mainline Linux kernels, eliminating the need for separate patches [75]. Control Groups provide a mechanism for partitioning sets of tasks into groups with specialized behavior [76]. These groups could for example have limited resources or specific associated CPUs.

3. MEASURING THE ENERGY EFFICIENCY

In this chapter the test environment and methodology will be introduced. The hardware used for conducting the measurements is introduced in Section 3.1. In Section 3.2 we describe how the testing procedure and analysis of results was conducted. The operating system choices are described in detail in Section 3.3. In the same section we also describe the configurations of the virtual machines. Finally in Section 3.4 we introduce the test applications and material. Regarding all the used software, if a configuration option is not mentioned, it was left at its default setting.

3.1 Test Environment Hardware

Testing was conducted on a Dell PowerEdge R410 server with two quad-core Intel Xeon E5520 processors and sixteen gigabytes of memory. Another server with two single-core Intel Xeon processors running at 2.80 gigahertz was used as a front-end computer.

Hyper-threading was disabled in the processors. This was done so that the physical cores could be reliably assigned to virtual machines as fully functional cores and not just logical ones. Intel Turbo Boost was enabled. Power management in motherboard BIOS was set to operating system managed. In practice, this means that the CPU clock frequency was fixed to 2.26 gigahertz for all cores except when Turbo Boost raised the operating frequency to 2.53 gigahertz. Some special tests were conducted with different BIOS power management, Turbo Boost and CPU clock frequency settings to see their impact on the results.

The R410 had a single 250 gigabyte hard disk drive. For VMware ESXi tests, a second hard disk drive of one terabyte was also installed. Input/output memory management unit (IOMMU) implementation Intel VT-d was not supported by the test system. Only the x86 hardware virtualization Intel VT-x was enabled. For network related tests, one part of the client-server pair was the front-end computer and the counterpart the R410. Network was routed through two D-Link DGS-1224T gigabit routers.

Power and energy usage data was collected with a *Watts up? PRO* meter with an accuracy of ± 1.5 percent plus three counts of the displayed value [77]. The meter was connected to the front-end computer via USB cable. The power meter was measuring the power consumption of the R410 server only. No display or other peripherals were connected to the R410. The small temperature changes in the server room might have caused the cooling fans of R410 to run at varying speeds, resulting in a nominal increase or decrease in power consumption. This, however, could not be measured or directly observed during the tests.

3.2 Test Methodology

The testing plan was the following: set up the R410 server with one virtualization technology and run various test applications multiple times measuring power and energy consumption. Then switch to another kind of virtualization technology and repeat the process. Finally, compare the results to those achieved with pure hardware without any virtualization. These tests run on non-virtualized environment are henceforth referred to as *the hardware tests*.

The process was automated with Bash shell scripts, whose functionality is illustrated in Figure 3.1. A script was started from the front-end computer. It commanded the R410 test server to start a virtual machine. When the virtual machine had booted, the front-end commanded the virtual machine to start the test application in question, again using shell scripts in the virtual machine. At the same time logging for the power meter was started. After the test run was finished, logging was stopped and the virtual machine was rebooted and the process was repeated. In the case of hardware tests, all of the above which concerns virtual machines was done directly on the R410.

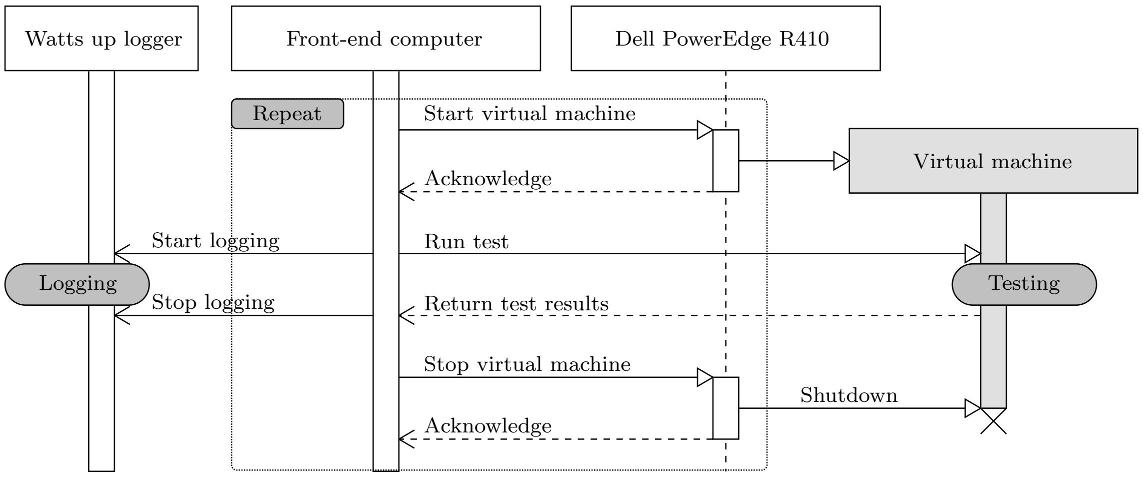

Figure 3.1: Sequence diagram of testing procedure. The sequence was repeated many times for each test to narrow down the error.

A number of test applications were used to simulate different kinds of tasks a computer might have to test the computer's resources inclusively. The tests can be roughly divided into three different categories: one focusing on processor performance, another focusing on network performance and the third focusing on disk input/output performance. All tests specific to only one category were synthetic: they do not represent a real life situation per se. A practical test type which tested all three types of stresses at the same time in a more realistic situation was also used.

The practical test type was a combined web server and database test. These tests were conducted using various virtual machine configurations to study on different virtualization technologies how the quality of service and energy efficiency behave when available resources are changed. Reliable quality of service is especially important for cloud service providers as service level agreements made with some parties may not have an impact on agreements made with other parties—too aggressive server consolidation can lead to performance loss [1]. The key statistic to assess the quality of service was chosen to be web server response time, as it is the one people first notice when browsing a website.

For all tests the output of the test application was recorded, giving performance characteristics for comparison. All along the instantaneous power usage and total energy consumption was measured every second. The results were then analyzed with custom Python scripts. An object-oriented parser collection was created for each test application, making it easier to add new test applications to the automated process in the future. Mean values and standard deviation (or the square root of the bias-corrected variance) were calculated from the results. A normal distribution was assumed between test runs and a 95 percent confidence interval (CI) was calculated as follows:

$$95\ \%\ \mathrm{CI} = \bar{x} \pm 1.96 * \frac{\sigma}{\sqrt{n}},$$

where $\bar{x}$ is the arithmetic sample mean, σ is the standard deviation, and n is the number of samples [78]. The results were automatically compared against hardware results, which mark the hundred percent reference points in all the resulting bar graphs. The 95 percent confidence intervals were also automatically plotted to the bar graphs. The automated result analysis sequence is illustrated in Figure 3.2. A sample Watts up power log is found in Appendix 1.

As one of the points was to study how hardware and power management features affect energy efficiency, some of the tests were conducted multiple times with different settings. The idea was to first run the tests and compare the results to get an overview of how different virtualization technologies compare to each other. After this comparison, we would pick one or two virtualization technologies, change the hardware settings and run some tests again to see the impact of different settings on the results.

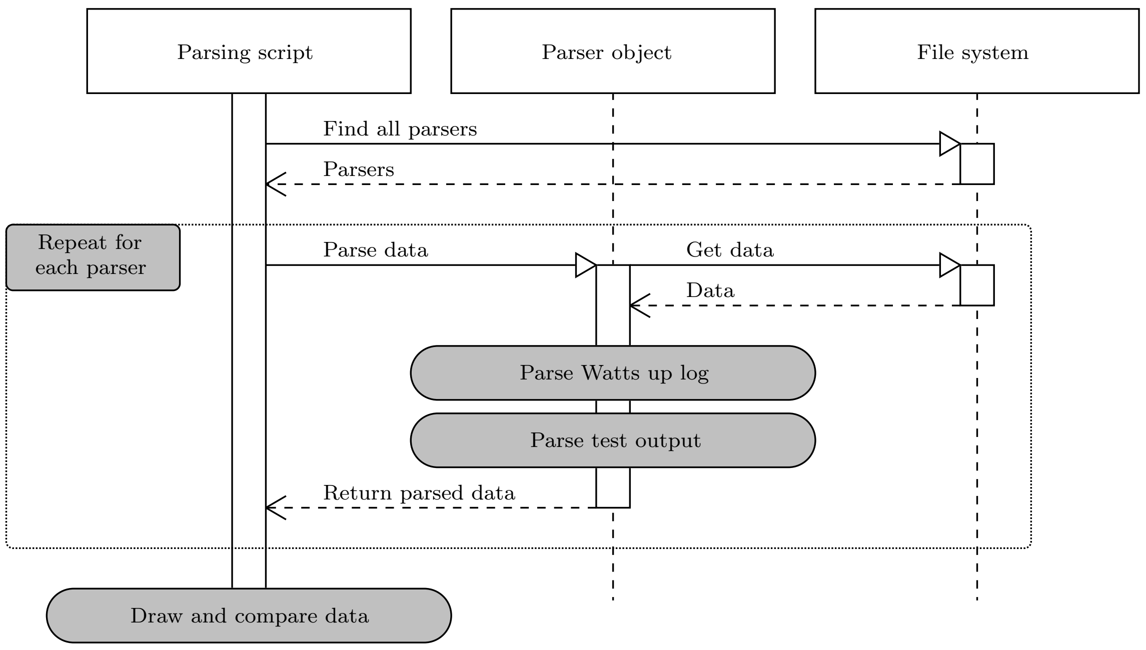

Figure 3.2: Parsing the results. For each test application a separate parser object, inherited from a common base parser object, was created.

3.3 Operating Systems and Virtual Machine Configurations

The operating system used in all the machines, be they virtual or real, was a default installation of 64-bit Ubuntu Server 10.04.3 LTS with the newest packages available as of August 26th 2011. The operating system was installed once. Copies of this installation were made to disk images and different hard disk partitions which were used by the virtualized environments. The same operating system used for hardware tests also served as the KVM host and was located in one partition. Xen domain zero was located in another separate partition. The virtual machine images which were used as guests were located in a third partition. All of the partitions and images had an ext4 file system. For VMware ESXi guests the same virtual machine images were used but run from a Virtual Machine File System (VMFS) partition located in a second hard disk drive. The ESXi itself was also installed in this second hard disk drive.

The virtual machine images were stored in raw format: an exact bit-for-bit copy of a real hard disk partition. Virtual machines were stored in image files instead of real partitions as it is frequent in the industry: for example in cloud computing, it is common to use a separate repository to store the virtual machine images for deployment [79].

The operating system kernel was Linux 3.0.0 compiled from mainline sources. The kernel was compiled with VirtIO drivers for the KVM guest. The VirtIO framework provides paravirtualized device drivers for network and disk devices [71]. Linux's paravirt_ops options were also enabled during compile time. They allow the kernel itself to be run paravirtualized on the Xen hypervisor.

Linux 3.0.0 was chosen mainly for two reasons. Firstly and more importantly, Linux 3.0.0 is the first kernel version to have a full Xen hypervisor support in it, allowing one to use the same kernel for both the Xen host operating system and the guest operating systems without kernel patches. As Linux 3.0.0 also has KVM support, it was possible to use the same kernel in all cases: in the non-virtualized environment, in the host operating system for KVM and Xen and in the virtual machines themselves. This eliminated the problem of different kernels having different performance, possibly affecting test results. In some cases, tests concerning Xen were also conducted using Linux 3.2.0 and 2.6.32.46 patched for Xen. The second reason for choosing Linux 3.0.0 is that it is relatively new and stable enough to be considered realistic for production use. In our tests we studied modern virtualization techniques; therefore a modern kernel with powerful virtualization capabilities readily available was a reasonable choice.

For web server tests, an installation of Scientific Linux CERN 5.6 was used inside a chroot jail. This was done to take advantage of an existing Invenio document repository installation, which is in production use at CERN. A test was conducted to assure the extra chroot in between the operating system and Invenio did not have any negative effects on test results. This test was a comparison between the base system and another chroot environment using a copy of the base system as the new root. No negative impact in using the chroot was measured.

The KVM version used was QEMU-KVM 0.15.0. Virtual machine configurations were given straight to the executable qemu-system-x86_64 as command line parameters. CPU type was set to match the host machine's CPU. For network interface controller, the paravirtualized VirtIO driver was used and the virtual controller was added to a network bridge configured on the host. Disk images were also used with the VirtIO driver. Cache mode was set to writethrough, which is the default. Some comparative tests were also conducted with cache mode set to writeback. A special comparative test using non-paravirtualized E1000 network device and virtual IDE hard disk drives was also conducted.

The Xen version used was 4.1.1. A separate core and 512 MB of memory was reserved for domain zero and the default Credit scheduler was used. Disk images were used as loop devices: a loop device in Linux is a nonphysical device node that makes a file accessible as a standard hard disk device would be. A special comparative test between the default Credit scheduler and newer Credit 2 scheduler was also conducted.

The ESXi version used was 5.0.0. Hypervisor power management settings were left at their default, balanced. Paravirtualized VMXNET3 network devices were used. Virtual machine image files were used as independent, persistent SCSI devices. The SCSI controller was paravirtualized and the SCSI bus was not shared between the virtual machines. As with KVM, a special comparative test using non-paravirtualized E1000 network device and virtual IDE hard disk drives was also conducted.

For hardware tests, the applications related to the test in question were executed with the taskset utility to set CPU affinity for the process. This was done in order to use the same amount of processing cores in hardware tests as in the virtual machine tests. Available system memory was limited with kernel boot parameter to be of the same size as in the virtual machine tests. When many virtual machines were run at the same time, comparison against hardware was made either against hardware matching one virtual machine's resources, or against hardware matching resources of the multiple virtual machines combined.

There were three virtual machine configurations. A configuration with four cores and eight gigabytes of system memory was the default. It was used in all of the synthetic benchmarks with the exception of I/O test, where the memory was lowered to two gigabytes to inhibit caching. Another configuration with only two cores and five gigabytes of system memory is referred to in the tests as a clone. Three copies of the same virtual machine image was made so that three identical virtual machines could be run at the same time. For a specific web server test, a configuration of one machine with six cores and fifteen gigabytes of memory was also used.

In most of the tests, power management setting in BIOS was set to operating system managed. This enabled the use of cpufreq governors, the dynamic CPU frequency and voltage scaling algorithms of Linux [80]. By default, the performance governor was used. It runs the CPU at maximum frequency all the time. In some power saving feature comparison tests, the conservative governor was used. It sets the CPU frequency depending on current usage.

3.4 Test Applications

Processor performance was measured with three benchmarks: Linpack, OpenMP and BurnInSSE, all three from Roy Longbottom's 64-bit benchmark collection [81]. The optimized version of Linpack was used. It is based on the popular LINPACK benchmark from the 1980s. LINPACK is a test which gives an estimate on raw computing speed in millions of floating point operations per second (MFLOPS); it is also used for ranking the 500 most powerful computer systems in the world [82]. Running Linpack is a simple way to find out if a virtualization technique poses any processor performance overhead. As the benchmark is quick to finish, it was run multiple times in sets of thirty consecutive runs to get reliable power consumption measurements. The Linpack test is single-threaded, however. A similar test but multithreaded, the OpenMP test was used to study a case where there are more computation threads then there are available processors. Processor power consumption behavior was studied by conducting ten minute burn-in runs with the BurnInSSE application. BurnInSSE stresses the processor to the maximum with a user-defined amount of threads. Tests were conducted using one, two and four threads.

Disk input and output performance was measured using Bonnie++ 1.96. Bonnie++ tests various I/O performance characteristics such as data read and write speed, seeking speed and various file metadata operation speeds [83]. These metadata operations include file creation and deletion and getting the file size or owner. The amount of files for Bonnie++'s small file creation test was 400. For large file test the file size was set to four gigabytes. For Bonnie++ tests, the amount of host operating system memory was limited to 2.5 gigabytes with kernel boot parameter, and the amount of guest operating system memory was limited to two gigabytes. For bare hardware test, a kernel boot parameter memory limit of two gigabytes was used. This memory limiting was necessary in order to prevent the large file test from caching any data. The tests were run five times.

Network performance was measured using iperf 2.0.5. It is a tool capable of measuring TCP bandwidth [84]. Three kinds of tests were run: one where the R410 test computer acted as a server, another where it was a client and a third where the R410 did a loopback test with itself. For client–server tests the front-end computer acted as the counterpart. Testing was done using four threads and a ten minute timespan to get reliable power consumption figures. TCP window size was sixteen kilobytes, the default. All three types of tests were carried out five times.

The practical test was a web server test based on Invenio document repository software suite v0.99.2. The document repository was run on Apache 2.2.3 web server and MySQL 5.0.77 database management system. The Apache web server was using its prefork module for multithreading. All these software were run on Scientific Linux CERN 5.6 inside a chroot jail. This is illustrated in Figure 3.3.

The front-end computer was used to send HTTP GET requests at a fixed rate of requests per second. The requests were similar to data which one would acquire from real log files of document repositories in use at CERN: for example the search terms included publications and persons. The total amount of requests sent corresponds to the average amount of requests the real production use repository gets in a day. The rate at which the requests were sent in the tests is much higher, however. This test generated all kinds of stresses to the test computer: network traffic, disk input/output and processor usage. The HTTP requesting and performance statistics recording was done using httperf 0.9.0, which is a tool to generate various HTTP workloads and measure service speed and latency [85]. The httperf application was

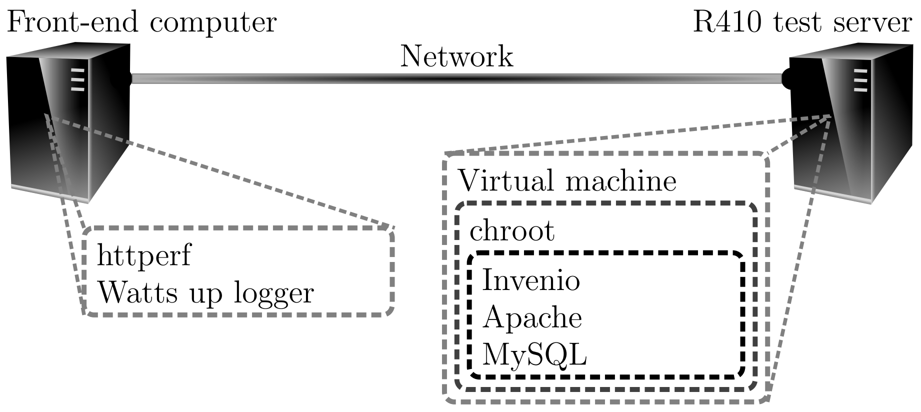

Figure 3.3: The web server test setup. Invenio, Apache and MySQL were running inside a chroot jail inside a virtual machine on the R410. In the front-end computer, HTTP workloads were generated with httperf and Watts up logger was recording the R410's power consumption. Routers in the network are not illustrated.

a slightly modified version which prints also the individual response times and not just the combined statistics. These response times were used to assess the quality of service. Response times of over ten seconds were discarded from the results and counted as errors. These abnormally high response times were mostly encountered at the beginning of the test, when the server was starting to fill its caches and start new threads.

The web server test was conducted using various virtual machine configurations. The default case used the same configuration as most of the other application tests. In Apache settings, maximum clients (MaxClients) were set to fifteen and the httperf request rate was ten requests per second. Another configuration, the *clone*, had less resources and consequently the MaxClients was set to eight while request rate was set to five requests per second. The clone configuration is the one which was used in case of multiple virtual machines. Tests were conducted with one, two and three virtual machines with the clone configuration. There were two reasons for choosing one to three virtual machines. First, the R410 server could not have handled any more without overlapping resources between the virtual machines—each virtual machine had a dedicated pair of cores. Second, in a report by the United States Environmental Protection Agency it was noted in an energy saving scenario overview that a physical server reduction ratio of approximately one to five was usually possible [86]. A ratio of three fits right in the range. A few tests were run with a special configuration with three times the resources of the clone configuration and consequently 24 MaxClients and a request rate of fifteen requests per second. When comparing the performance with three virtual machines against hardware, this special configuration was the one used for hardware. These configurations are summarized in Table 3.1. All the tests were carried out ten times for each configuration.

Table 3.1: Virtual machine configurations for the web server test. Memory size is in gigabytes (GB) and request rate in requests per second.

Setup type	**Cores**	**Memory (GB)**	**MaxClients**	**Request rate**
Default	4	8	15	10
Clone	2	5	8	5
Special	6	15	24	15

To summarize the used test applications and relevant software, they are listed with their version numbers in Table 3.2. A sample output from all the test applications is found in Appendix 1.

Table 3.2: All the relevant software with their versions and purpose.

Software	**Version(s)**	**Purpose**
Apache	2.2.3	Web server
Bonnie++	1.96	I/O performance
BurnInSSE	64-bit	Multithreaded CPU stress testing
ESXi	5.0.0	Hypervisor
httperf	0.9.0	Web server performance
Invenio	v0.99.2	Document repository
iperf	2.0.5	Network performance
Linpack	64-bit optimized	Single-threaded CPU speed testing
Linux	3.0.0, 2.6.32.46, 3.2.0	Kernel
MySQL	5.0.77	Database
OpenMP	64-bit	Multithreaded CPU speed testing
QEMU-KVM	0.15.0	Hypervisor
Scientific Linux CERN	5.6	Operating system inside chroot jail
Ubuntu	10.04.3 LTS 64-bit	Operating system
Xen	4.1.1	Hypervisor

In addition to the application tests above, also the idle power consumption was measured for each test environment. Idle power consumption was recorded with multiple fifteen minute measurement periods. Measurements were made for setups of one and three virtual machines and with a variety of hardware settings.

4. RESULTS

Test results are presented and discussed in this chapter. In Section 4.1 we introduce the results from synthetic tests. They include for example disk input/output, network and processor performance tests. Idle power consumption measurements are also discussed in the section. In Section 4.2 we present the results from the more practical web server tests and assess the quality of service. In Section 4.3 we present the results from various special test cases. These include for example the effect of Turbo Boost and different kinds of power management settings.

In the bar graph style figures in this chapter, the black lines over bars depict the 95 % certainty intervals. The *hardware* environment type in legends refers to the non-virtualized test environment. In all of the tests concerning ESXi, a second hard disk drive was installed in the system. The measured impact of the second hard disk drive on power consumption is presented in the idle test results in Section 4.1.

4.1 Synthetic Tests

The first results in Table 4.1 are from idle power consumption measurements. The test is synthetic in the sense that usually servers have at least some load [9].

Table 4.1: Idle power consumptions. The columns are: environment type, number of hard disk drives (HDDs), number of virtual machines (VMs), mean power consumption ($\bar{P}$), $\bar{P}$ relative to one HDD hardware result (% of HW), standard deviation (σ) and 95 percent confidence interval (95 %). Xen 2.6 denotes Xen with Linux 2.6.32.46.

Type	**HDDs**	**VMs**	$\bar{P}$ **(W)**	**% of HW**	σ **(W)**	**95 % (W)**
Hardware	1	-	75.4	100.0	0.05	±0.05
	2	-	82.1	108.9	0.04	±0.05
KVM	1	1	77.7	103.1	0.31	±0.35
		3	80.1	106.2	0.47	±0.51
ESXi	2	1	139.6	185.1	0.11	±0.12
		3	139.7	185.2	0.08	±0.09
Xen	1	1	134.3	178.1	0.24	±0.19
		3	134.0	177.7	0.61	±0.38
Xen 2.6	1	3	103.1	136.7	0.39	±0.45
Special	1	-	103.7	137.4	1.20	±0.96

As can be seen in Table 4.1, the idle, non-virtualized system consumes less than eighty watts of power. Adding a second hard disk drive raises the consumption by almost seven watts. This should be noted in all measurements concerning ESXi, as its test setup had the second hard disk drive installed.

There is very little difference in idle power consumption between running one and three virtual machines. KVM consumed only little more power than hardware, but Xen consumed almost eighty percent more. Subtracting the second hard disk drive consumption from the ESXi idle results, we see that ESXi consumed similar amount of power as Xen. We also tested Xen with kernel version 2.6.32.46 and 3.2.0 in addition to the 3.0.0, which was our default. Using the older kernel yielded a power consumption level of approximately hundred watts, while the newer kernel had a power consumption similar to the 3.0.0. Therefore with newer kernels, which have the Xen hypervisor functionality without any patches, there seems to be something different in the way Xen behaves.

The special environment at the bottom of the table is the single hard disk drive hardware environment but with a USB keyboard plugged in. In the hardware there exist some subsystems, which are not active or consuming power without any peripherals. The USB root hub is one of these devices. On hardware and KVM, plugging in a USB device results in vastly increased idle power consumption. With ESXi and newer Xen the power consumption added by the USB device is on the level of few watts as expected. This is also the case in the other environments if there already is system load. This indicates a big difference between the power management behavior of bare-metal hypervisors Xen and ESXi versus KVM, which is a standard operating system kernel turned into a hypervisor.

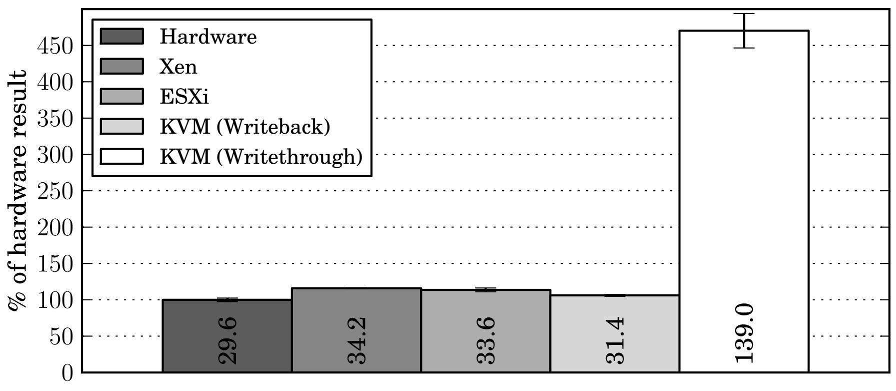

Figure 4.1: Energy consumed by a single Bonnie++ run in watt hours. Only small differences in total energy consumption can be seen except with KVM between different cache modes.

We measured disk input/output energy efficiency by running a set of disk operations with Bonnie++. The results are shown in Figure 4.1. Xen uses slightly more energy compared to hardware to finish these operations. ESXi uses a bit less energy than Xen even with its second hard disk drive, although it was also using a different file system. With KVM the situation is different. When using the default writethrough cache mode, KVM uses over 450 % as much energy as hardware to complete the operations. The energy consumption is high because the test takes much more time to finish. In the writethrough case, ninety percent of the time was spent doing the small file test section of Bonnie++. Switching to writeback cache mode, KVM's results are very close to hardware level. Writeback cache mode writes to a cache, which is written to permanent storage only just before the cache is replaced. This cache mode is not safe for production use and is recommended mainly for testing purposes.

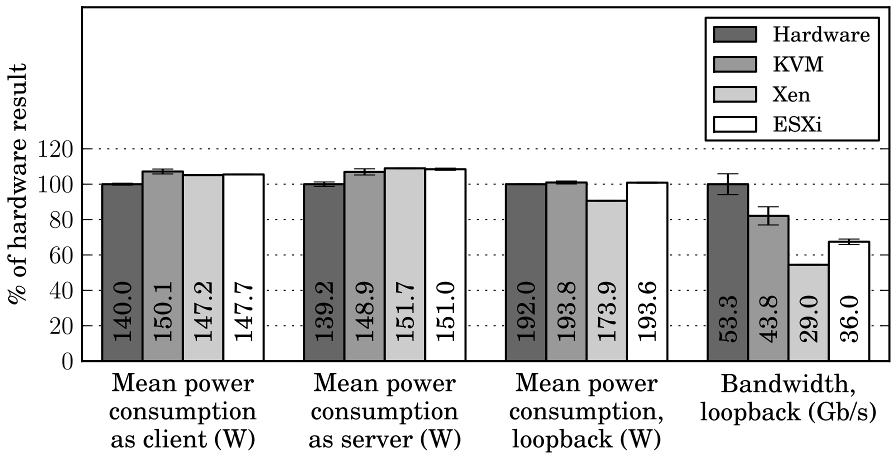

Figure 4.2: Results for iperf network performance test. Bandwidth is in gigabits per second.

In Figure 4.2 we have network performance results from iperf test. Bandwidth in both client and server cases was approximately 940 megabits per second for all types of virtualization technologies. As the routers in the network were gigabit routers, taking into account the small TCP header overheads one can say the test equipment was working at its theoretical speed limits.

ESXi with a fully virtualized network device achieved the gigabit ethernet speed, but for some reason, with our Ubuntu installation the paravirtual device functioned only at a hundred megabits per second. In our tests, however, bandwidth was never the issue. That is why in the iperf results we only present the mean power consumption for client and server cases. The results in Figure 4.2 are for the paravirtualized network devices. Both as the client and as the server, all of the virtualized environments had a bit bigger power consumption compared to the hardware, but similar to each other. Adjusting for the second hard disk drive, ESXi has a little lower power consumption than KVM or Xen.

In the loopback test the situation is different. The bandwidth is less with each hypervisor than with pure hardware. Important to note is how Xen consumes less power than the others but also suffers from limited loopback bandwidth. This implicates that Xen is not working as fast as would be possible if the computer's hardware was the only limiting factor. Loopback bandwidth with ESXi was between Xen and KVM. The absolute values are however several gigabytes per second, a bandwidth covering almost all real life situations.

Assessing qualitatively, KVM had troubles in the iperf tests. The KVM virtual machine's network seemed to cease functioning quite often during the tests, preventing any remote connections to the virtual machine. KVM worked reliably only in the loopback test. Bare hardware, Xen, and ESXi had no problems with reliability.

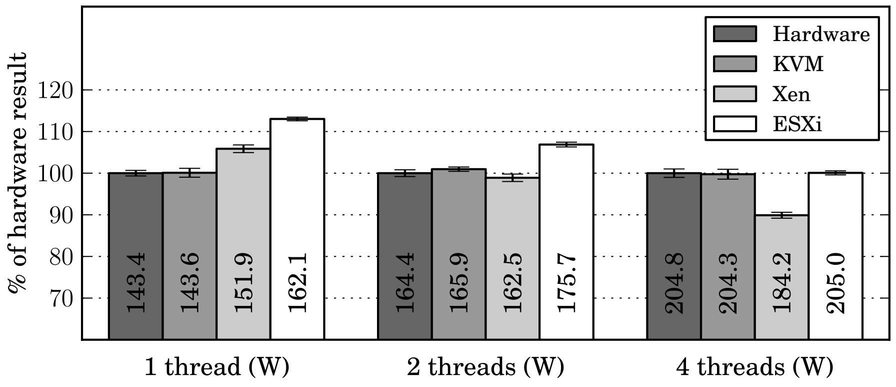

Figure 4.3: Mean power consumption in BurnInSSE with one, two and four computing threads stressing the processor.

In Figure 4.3 are the measured power consumptions under full CPU load for one, two and four computing threads of BurnInSSE application. With one thread, KVM and hardware consume the same amount of power, while Xen consumes six percent more than hardware. Adjusting for the second hard disk drive, ESXi consumes marginally more power than Xen. With two threads all contestants consume approximately similar amount of power with the exception of ESXi's slightly higher consumption. With four threads Xen uses less power than KVM and hardware. The BurnInSSE benchmark only adds load to the CPU; it does not produce any results per se. To explain the phenomenon in Xen's power consumption, we must also consider the Linpack results in Figure 4.4.

From the Linpack results in Figure 4.4 we notice that Xen consumes more power than hardware, just as in BurnInSSE with only one thread. We also note that the computing speed is below ninety percent of hardware speed and the test takes more time to finish. Hence the energy consumption difference is relatively bigger than the power consumption difference compared to hardware. Using Xen, the CPU is not running at its full speed—Xen has a systematic overhead in power consumption. This is why in Figure 4.3 Xen consumes less power with four threads: the processor

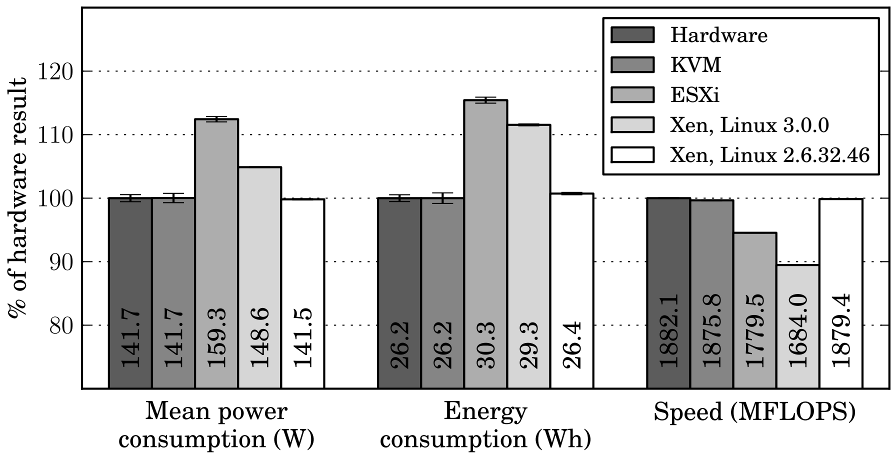

Figure 4.4: Energy consumption characteristics and achieved computing speed in Linpack. Speed is in millions of floating point operations per second.

is not working at its full potential and the systematic power consumption overhead is overshadowed by the power required by the four computing threads. With the older kernel, Xen's performance in Linpack is on par with hardware.

By further studying the Linpack results, we notice that the ratio between the hardware and Xen computing speed results is the same as the ratio between the turbo clock frequency and nominal clock frequency of the Intel E5520 processor in our test computer. It can thus be concluded that the bad performance of Xen in Linpack test is due to lack of Turbo Boost, which has not turned on because Xen is keeping the whole processor busy to some extent.

KVM's Linpack performance is on par with the hardware, but ESXi's speed is between Xen and hardware. This means that with ESXi, Turbo Boost is active but not working at full speed. Thus ESXi also has some overhead in processor usage. The higher power consumption of ESXi compared to Xen in Figure 4.3 and in Figure 4.4 is also explained by the presence of Turbo Boost, after adjusting for the second hard disk drive of ESXi. These findings about Xen and ESXi are in line with those observable in the idle test results in Table 4.1.

4.2 Web Server Tests

Httperf was used for testing the performance of virtualized web servers. First, test runs were done using the same single virtual machine as with the synthetic tests. The results of the tests for this quad-core virtual machine are presented in Figure 4.5.

For the single virtual machine setup, ESXi offers native level performance with the cost of increased power consumption compared to hardware. KVM consumes only little more energy than hardware, but transfer times are worse. Xen has both

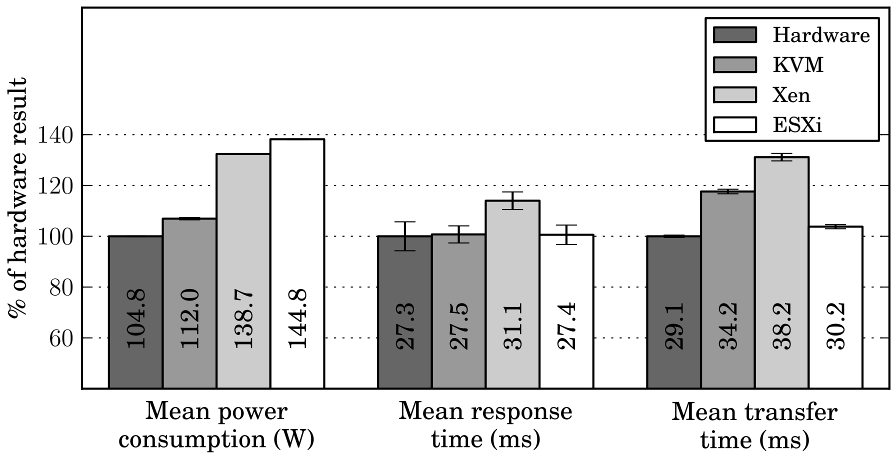

Figure 4.5: Httperf power consumption and performance results for one virtual machine.

increased response and transfer times in addition to a power consumption of ESXi level, after taking into account the second hard disk drive of ESXi. Even though the relative changes in response and transfer times are clear in the figure, the differences in absolute values are small. Depending on the type of web service, a human user would not notice if the service was a few milliseconds slower.

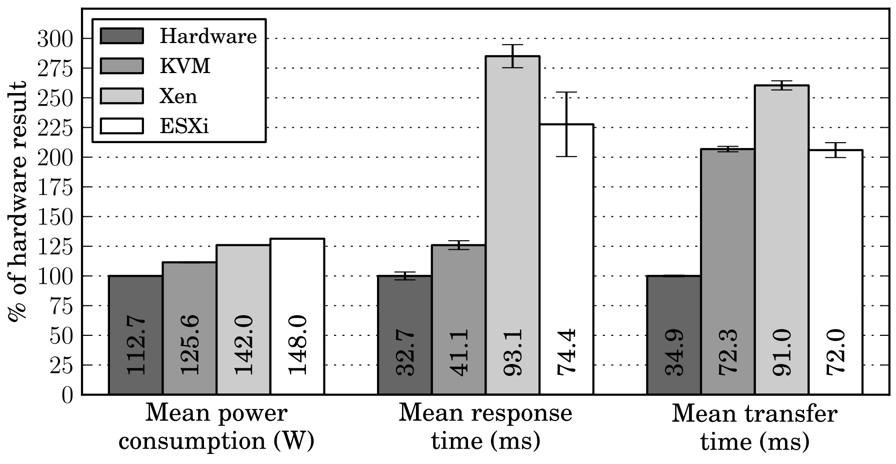

Figure 4.6: Httperf results for three virtual machines.

In Figure 4.6 are the results when using three dual-core virtual machines. The reference hardware setup had six cores. Power consumption figures are similar to the one virtual machine case in Figure 4.5, to the extent that for Xen and ESXi the power consumptions have not risen that much. They are still clearly higher than with hardware or KVM, however. ESXi and Xen are again on par with power consumption after adjusting for the second hard disk drive of ESXi. Still comparing to the one virtual machine case, response times have risen a little for KVM and a lot for Xen. ESXi's response times have radically increased from native performance level to nearly Xen's level. The response times have also not been as stable as with other technologies. Transfer times have increased a lot for each hypervisor.

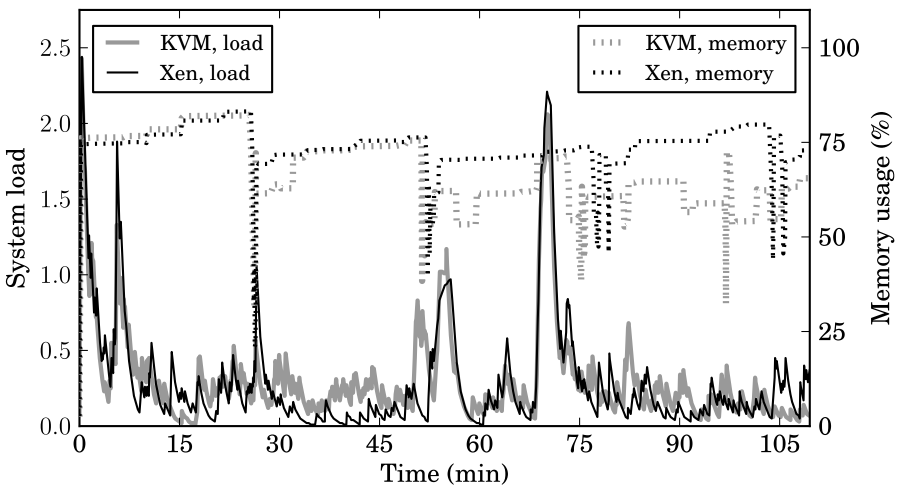

Figure 4.7: A virtual machine's system load and memory usage during a httperf run.

In Figure 4.7 we have plots of a virtual machine's system load and memory usage during a httperf run. Data in Figure 4.6 consists of runs like the one in Figure 4.7. As can be seen, system memory usage never reaches hundred percent. System loads, acquired with the uptime command from the last one minute period, are also most of the time well below two, which is the number of cores the virtual machine had. In the figure we have plotted only KVM's and Xen's results. They are very similar to each other. The hardware and ESXi results were similar as well and thus not displayed. As the results in all four cases were very similar, differences in Figure 4.6 originated from the hypervisors, not from what was happening inside the virtual machines.

There is a relatively high load level in the beginning of the test, because for example Apache and Invenio are starting their processing threads. This is why, on average, the first responses take more time than the rest. We excluded up to the first three percent of responses and noticed that the effect on mean response times was similar with and without virtualization. The effect was relatively smaller with greater amounts of virtual machines. In practice, however, this effect was a small one and therefore the results shown are calculated from all responses, including the first ones.

In Figure 4.8 is a comparison of httperf performance for KVM using one, two and three virtual machines. Power consumption increases linearly with each virtual machine consuming approximately thirteen watts more. Response times take a big hit when adding a second virtual machine, but the addition of a third virtual machine has only a small effect. With transfer times the situation is the opposite, with the

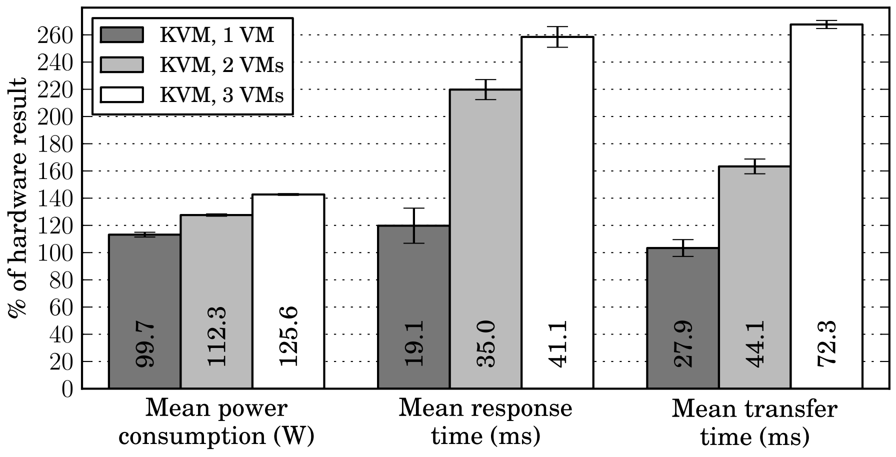

Figure 4.8: Httperf results for KVM using different amount of virtual machines. Reference is the hardware setup corresponding to one virtual machine.

addition of the second virtual machine having a smaller effect than the addition of the third.

Figure 4.8 illustrates the benefits of server consolidation very clearly. With only a little higher power consumption, one gets double or triple the work done without too much of an impact on performance. If the response and transfer times fit in the customers' service level agreements, one could consolidate the servers and save energy. In our test case, the savings would be almost sixty percent compared to the option of running each server in their own dedicated hardware.

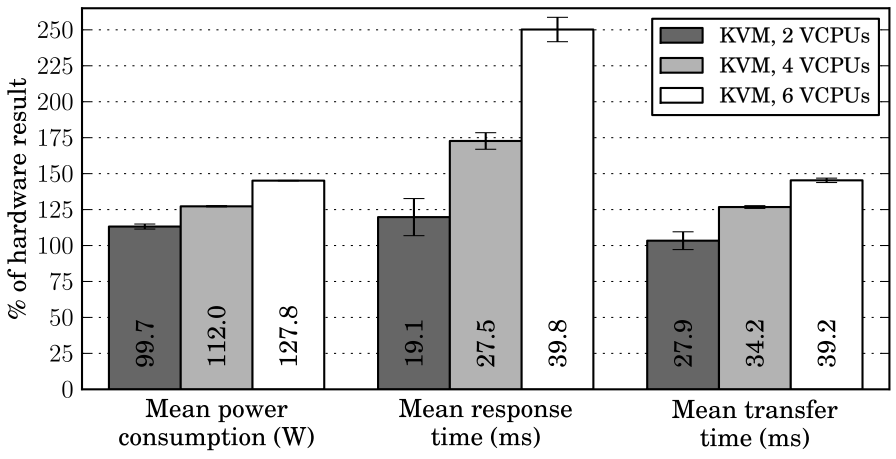

Figure 4.9: Httperf results for KVM using different virtual machine resources. VCPUs denote the amount of virtual processors.

We also tested a similar situation with one virtual machine using three different amounts of resources. As resources were expanded, the load was increased accordingly. The results are shown in Figure 4.9. Power consumption grows linearly again and is very close to that in Figure 4.8. Using just one virtual machine, also the response and transfer time growths follow a somewhat linear fashion. The overhead of using multiple virtual machines instead of just one with great resources is thus observable, but not that big. KVM copes well in both cases.

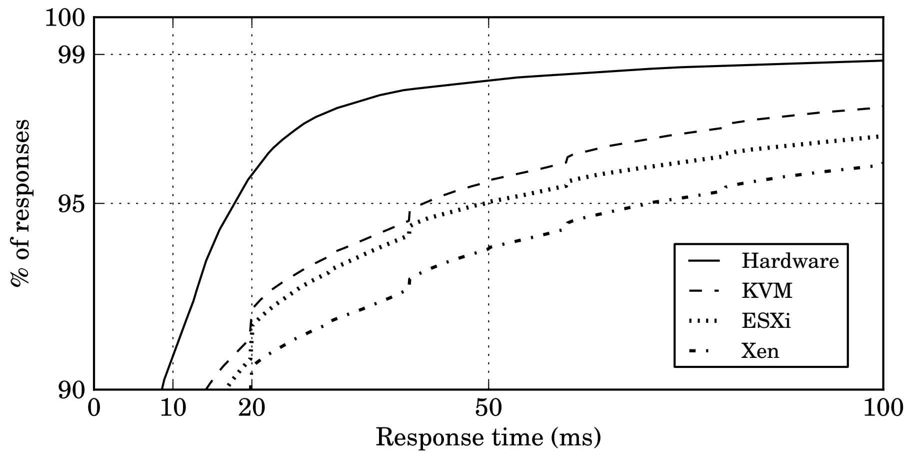

Figure 4.10: Quality of service curves for the three virtual machine setup. The curves depict how big a percentage of response times were below a given value during the httperf test runs.

Moving on to the quality of service assessment, in Figure 4.10 we see how big a percentage of response times were below a given value. Illustrated are the results for the three virtual machine case. For all virtualized environments, at least ninety percent of response times were under twenty milliseconds. Using bare hardware the response times are better and in twenty milliseconds the 95 percent coverage level is reached. Depending on service type and service level agreements, this kind of statistic could be used to evaluate whether a virtualization technology is suitable for the situation. For example in our case, if 95 percent of response times should be under fifty milliseconds, KVM and ESXi would be suitable technologies. The corresponding power consumption figures for these statistics are in Figure 4.6.

In Figure 4.11 we have the quality of service statistics for KVM using different amounts of virtual machines. From how the number of virtual machines affects the shape of response time curves, one is able to predict how the quality of service would be affected should more virtual machines be added. Hence, this type of figure could be used as a rough estimate whether adding an extra virtual machine would still result in satisfied service level agreements. The corresponding power consumption figures are in Figure 4.8.

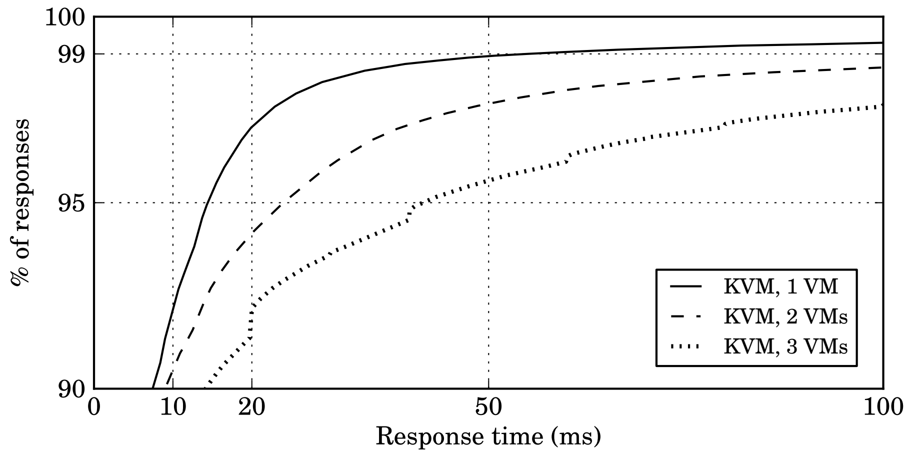

Figure 4.11: Quality of service curves for KVM using different amount of virtual machines. The curves depict how big a percentage of response times were below a given value during the httperf test runs.

4.3 Special Test Cases

In this section we analyze special test cases where the basic test environment setup was adjusted in some manner. One of the adjustments is changing the power management settings. In system BIOS, one can choose between *system managed* and *operating system managed*. The former is the default and leaves everything for the hardware, while the latter enables using CPU governors to adjust processor operating frequency and voltage. In Table 4.2 we have the server's idle power consumption using both of these power management settings.

Table 4.2: Hardware idle power consumption using various power management settings. In the column labels, $\bar{P}$ denotes the mean power consumption, σ the standard deviation and 95 % the 95 percent confidence interval.

Power management setting	**Governor**	$\bar{P}$ **(W)**	σ **(W)**	**95 % (W)**
Operating system managed	Performance	75.4	0.05	±0.05
	Conservative	75.1	0.19	±0.22
System managed	-	75.8	0.11	±0.12

As can be seen, there is virtually no difference in power consumption level with any of the used power management settings. The explanation is that modern processors, for example Intel Xeons as in our test server, take advantage of processor power states—the processors rely not only on frequency and voltage control to save energy. When the system is idle, no load is on the processor and the processor is set to a more energy saving power state. Hence, even if the operating frequency and voltage is lower with the conservative governor, the power consumption while idle is the same with performance governor because the processor is sleeping.

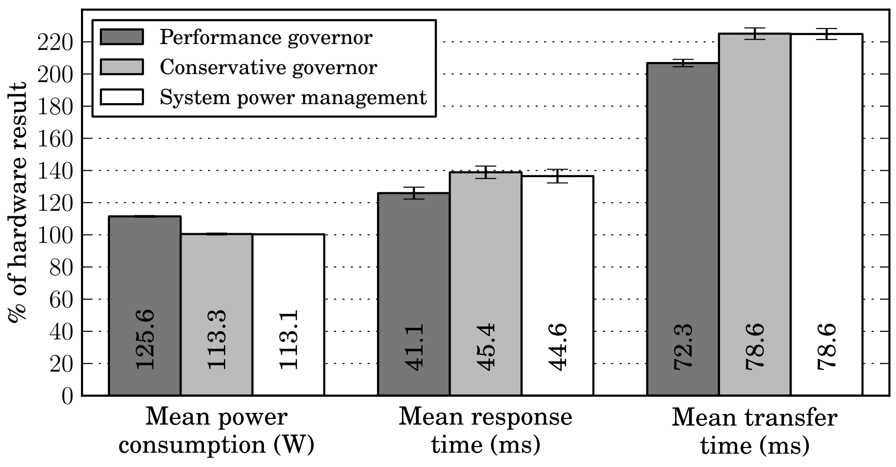

Figure 4.12: Httperf results for three KVM virtual machines using different power management settings.

In Figure 4.12 we have httperf results for three KVM virtual machines with various power management settings. Using the conservative governor saves approximately ten percent of energy, while performance is in turn almost ten percent worse. The absolute differences are, however, again very small. Using the default system power management option the power consumption and performance is on the conservative governor level.

By changing the power management settings, one may either enhance performance or energy efficiency. With our test server using Intel Xeon processors, the energy savings are readily achievable by the default power management settings; with another server without good system power management, switching to the conservative CPU governor would result in better energy efficiency. This is especially the case if the processor's energy saving features were not based on processor power states but on processor operating frequencies.

These improvements could only be achieved with KVM, however. A similar test as in Figure 4.12 with Xen resulted in no energy savings at all using the system power management. Xen's virtual machines are also unable to take advantage of CPU governors. Forcing the Xen privileged domain to use the conservative governor results in power savings but very bad performance: the operating frequencies are never raised as the frequency control logic in the privileged domain's Linux is unaware of the system loads in the unprivileged domain guests. Similarly, on our system no power management settings resulted in better energy efficiency with ESXi guests.

In Table 4.3 we have power consumption statistics with and without Turbo Boost for the httperf test with three virtual machines on KVM hypervisor. With Turbo Boost disabled, over seven percent of energy is saved. What is important is that no observable degradation in performance was measured when Turbo Boost was disabled. Consequently, the response and transfer times are omitted from the table.

Table 4.3: Httperf power consumption for three KVM virtual machines with and without Turbo Boost. In the column labels, $\bar{P}$ denotes the mean power consumption, σ the standard deviation and 95 % the 95 percent confidence interval.

Turbo Boost	$\bar{P}$ **(W)**	σ **(W)**	**95 % (W)**
Enabled	125.6	0.90	±0.56
Disabled	116.3	0.79	±0.49

In Figure 4.13 are the results of an iperf test for KVM with and without Turbo Boost. Disabling Turbo Boost results in energy savings of five to six percent when running the server as iperf client or iperf server. In loopback mode the impact on power consumption is over eight percent. Some performance loss is observable, but statistically insignificant. These results indicate that the mediocre performance of Xen in the iperf test, whose results are in Figure 4.2, are not due to the lack of Turbo Boost when using Xen.

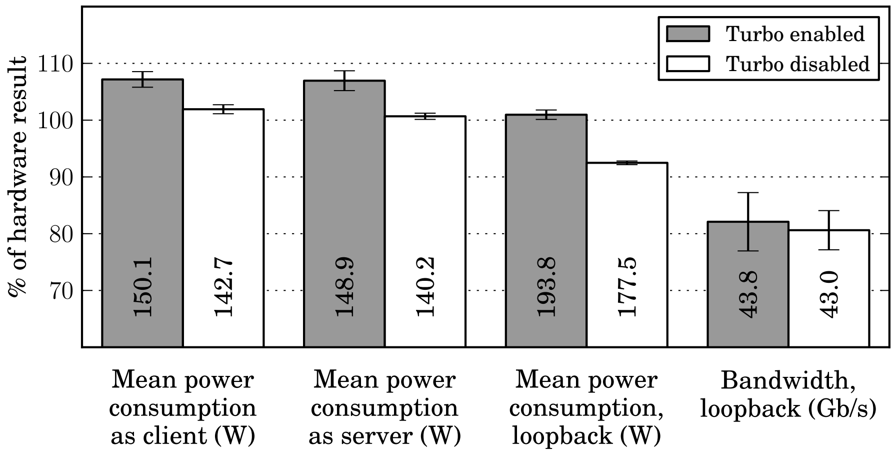

Figure 4.13: Results for KVM with and without Turbo Boost in iperf test. Bandwidth is in gigabits per second.

From Table 4.3 and Figure 4.13 we see that albeit Turbo Boost can improve performance in processing power critical computing tasks, using it in a web server only impairs energy efficiency.

In Figure 4.14 are the results for the three virtual machine httperf test with ESXi. The two cases shown are the paravirtualized case, which was the default, and the non-paravirtualized case with fully virtualized network adapter and hard

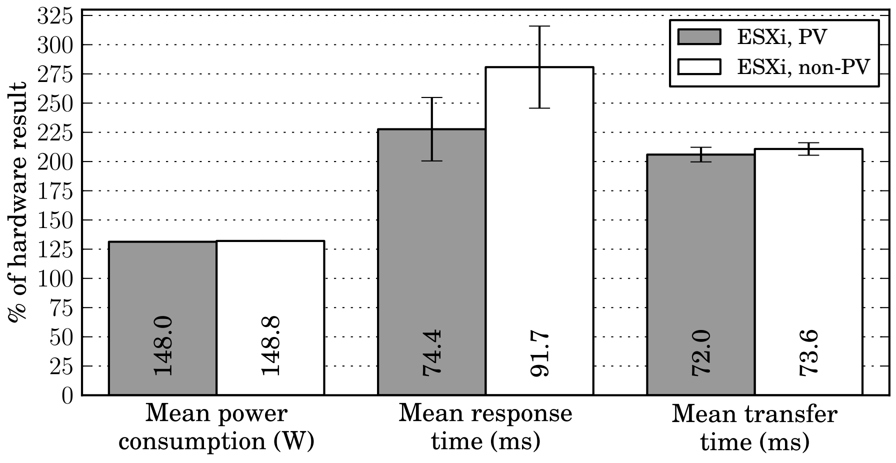

Figure 4.14: Httperf results for three ESXi virtual machines with and without paravirtualization.

disk drives. As can be seen, in this case the paravirtualization only affects response time. The results also confirm that our tests did not require much bandwidth: on our installation, the paravirtualized device was working at a hundred megabits per second as opposed to the one gigabit per second speed of the fully virtualized device, but there were no differences in transfer speeds.

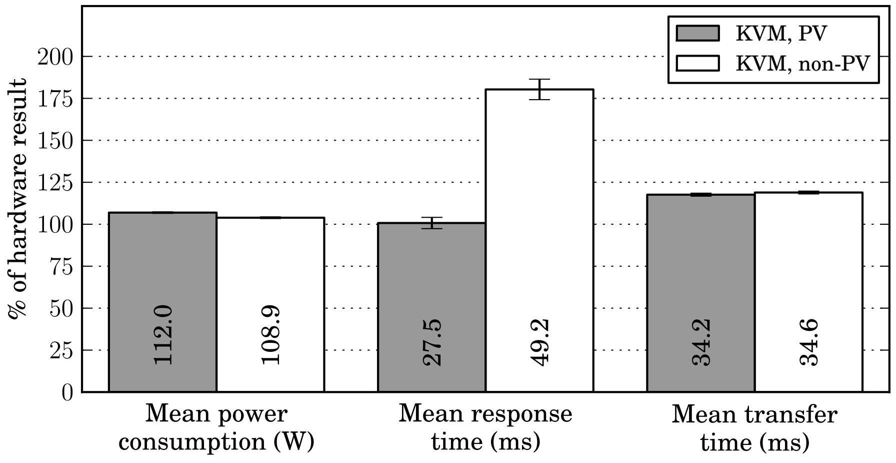

Figure 4.15: Httperf results for a KVM virtual machine with and without paravirtualization.

Figure 4.15 shows the effect of paravirtualization in the httperf test using KVM. In the non-paravirtualized case, the network adapter and hard disk drives were fully virtualized. As with ESXi in Figure 4.14, paravirtualization mainly affects the response times. Interestingly, power consumption is marginally smaller when using full virtualization.

In Figure 4.16 is illustrated the effect of moving the Scientific Linux CERN (SLC)

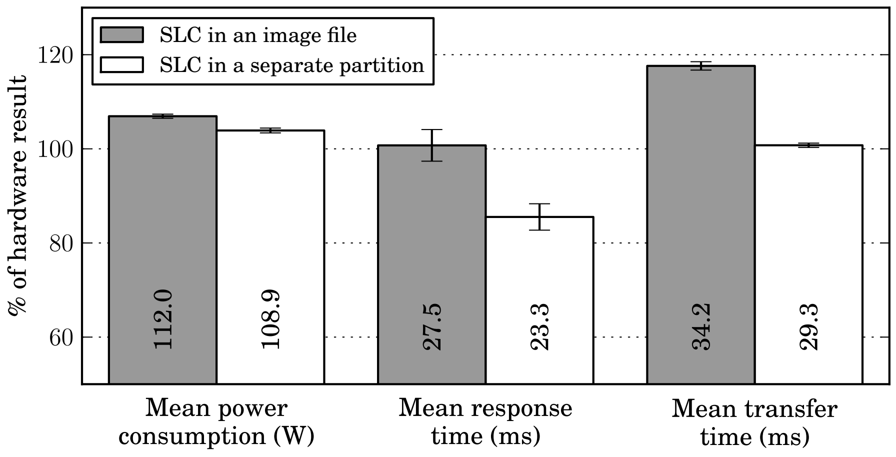

Figure 4.16: Httperf results for KVM with the Scientific Linux CERN installation in an image file and in a separate partition.

installation from an image file to a separate partition in the httperf test with one KVM virtual machine. In the httperf tests, SLC was inside a chroot jail and hosted the web services. Using a separate partition for the SLC results in marginally better energy efficiency, but clearly improved response and transfer times. For quality of service critical situations, using a dedicated partition might thus give better results.

In Table 4.4 we have power and energy consumption results from four sets of OpenMP tests with Xen. OpenMP measures CPU speed by using as many threads as there are processors in the system. We ran the OpenMP test with three virtual machines each using either two or four virtual processors. In case of two processors, the hardware has enough cores to dedicate one for each virtual processor. No difference between the performance of default Credit scheduler and the newer Credit 2 scheduler is observed.

Table 4.4: Effect of different schedulers on Xen's performance in OpenMP test. VCPUs denotes the total amount of virtual processors used by the virtual machines, $\bar{P}$ the mean power consumption, σ the standard deviation and $\bar{E}$ the total energy consumption.

VCPUs	**Scheduler**	$\bar{P}$ **(W)**	σ_{power} **(W)**	$\bar{E}$ **(Wh)**	σ_{energy} **(Wh)**
3×2	Credit	181.5	0.33	19.5	0.1
	Credit 2	181.2	0.31	19.6	0.0
3×4	Credit	161.3	0.39	245.0	50.9
	Credit 2	190.1	0.71	18.7	1.2

With four virtual processors per virtual machine the situation is different. As there are a total of twelve virtual processors and the hardware only has eight cores, Xen's scheduler has to share processor resources between the virtual machines. As can be seen from the table, using the default Credit scheduler the energy consumption is thirteen-fold compared to the results achieved with Credit 2 scheduler. The mean power consumption is even lower than in the case of dual-core virtual machines. These results originate from the long time it takes to finish the test with the hindered performance of Credit scheduler. The energy consumption also varies a lot between test runs. With Credit 2 scheduler the performance is rational. Albeit a synthetic test, this comparison shows that one should be cautious with allocating resources for Xen virtual machines when using the default Credit scheduler. Running the OpenMP tests with KVM and ESXi resulted in expected behavior with no scheduling problems.

5. CONCLUSIONS

Measuring energy efficiency for a server requires knowledge of performance and power consumption characteristics. We conducted sets of tests while measuring power consumption. From the test results we assessed how virtualization affects performance and consequently energy efficiency. In addition to bare hardware, these tests were run using three hypervisors: KVM, Xen, and ESXi. We also tested various configurations to find out what should one generally keep in mind when building a virtualized environment with focus on energy efficiency.

KVM offers very good energy efficiency. In most tests, KVM had only little virtualization power consumption overhead, and performance was among the best. Raw computing speed was on bare hardware level and in web server tests the performance overhead was predictable when the number of virtual machines or virtual machine resources were modified. Hardware energy saving features were fully functional using KVM. For example we were able to use CPU governors to set CPU clock frequency from the operating system.

KVM had trouble only in Bonnie++ disk input/output and iperf network bandwidth tests. The slow disk operations did not seem to affect practical performance, however. The problems in iperf tests were not quantitative, but qualitative. If these problems were fixed, KVM would be an excellent choice, and in most situations already is. For example Linux distribution vendor Red Hat has already adopted to use KVM as its main virtualization solution [87].

LXC is not yet mature enough for production use, and we were unable to conduct tests using it. LXC has a good premise, however. As LXC uses features found in the mainline Linux kernel, LXC will be a noteworthy option in the future as older container-based virtualization technologies become obsolete.

Xen was a troublesome contender with respect to both performance and power consumption. In most of the tests, its performance was the worst among tested technologies. With the newer version 3.0.0 and 3.2.0 Linux kernels, there existed a base overhead in power consumption and obviously in processor usage as Turbo Boost refused to activate, impairing performance in computing power test. An older kernel from version 2.6 series did not have such problems. Power consumption with the older kernel was still greater than with KVM. Energy saving features like dynamic CPU clock frequencies could not be used with Xen.

Xen seems to be living a transitional phase in general. For example, in our tests it was noted how the default Credit scheduler of Xen had much trouble with cases where the number of virtual processors was greater than the amount of physical cores. This scheduler has received bad test results also in the past [15, 28]. The Xen developers have also stated that the current default scheduler has problems in some cases, and the next version, Credit 2, should address these issues [88].

ESXi, or the VMware vSphere Hypervisor, had power consumption levels similar to Xen. Hardware energy saving features did not work on our test system with ESXi. This would require further studying on different hardware, as ESXi should have host power management capabilities [89]. Unlike with Xen, Turbo Boost could activate. It could not work at its full potential, however, indicating some sort of processor usage overhead. With respect to performance, in most tests ESXi had good performance just behind KVM. In a single virtual machine web server test, ESXi achieved bare hardware level performance. This shows that native level performance is achievable with virtual computer even in complex situations where all kinds of system resources are utilized at the same time, as opposed to for example a simple, synthetic CPU benchmark.

In our web server tests, using Turbo Boost resulted in increased power consumption with no observable performance benefits. When setting up an energy efficient system, one should therefore assess the benefits of Turbo Boost to its downsides. Using paravirtualization yielded no improvement on power consumption. Performance using paravirtualization was marginally better with both KVM and ESXi, the two hypervisors which were chosen for the web server paravirtualization tests. The effects of paravirtualization on energy efficiency should be studied with wider sets of tests to find out if it really does not have that much meaning, as using full virtualization might have its benefits, for example greater flexibility. These other benefits could then lead to better overall energy efficiency.

Another point worth closer studying is how the bare-metal hypervisors ESXi and Xen consumed much more power than KVM, and similar amount of power to each other. This was especially true on lighter system loads. The difference obviously lies in the way the operating system handles some hardware subsystems as could be seen in system power consumption figures when plugging in a USB keyboard to activate a USB root hub. What should be studied is what these hardware subsystems exactly are, why are they not in energy saving mode with bare-metal hypervisors, and is this phenomenon observable when using different server hardware.

From the test results it is clear that by using virtualization for server consolidation one achieves great energy savings independent of the virtualization technology used. As we were able to use the same virtual machine images with different hypervisors with practically no modifications to the images, changing the hypervisor according to situation is also a realistic possibility. Using different technologies might offer other benefits, such as wider operating system support, better migration capabilities, better isolation possibilities, or simply support for the virtualization solution currently in use. There are differences between technologies in both performance and power consumption, but the resulting energy efficiency is a combination of choosing the correct virtualization technology for the situation and tuning the environment to best suit the technology. To ease maintenance challenges in such situations, there exists tools like libvirt [90] for managing multiple virtualization technologies.

Virtualization has proven to be a succesful technique to achieve the flexibility and cost-effectiveness required to build modern information technology infrastructures. Commercially, virtualization has been a success, but also the open source world offers interesting and capable virtualization solutions. KVM shows that Linux, the same kernel that is used in computers ranging from servers to personal computers and mobile devices, can be turned into an energy efficient hypervisor ready for production use. Computing is greener on the virtualized side of the fence, but nowadays passing the fence is truly possible for everyone.

BIBLIOGRAPHY

[1] A. Beloglazov and R. Buyya, “Energy Efficient Allocation of Virtual Machines in Cloud Data Centers,” in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, may 2010, pp. 577 –578.

[2] Amazon Web Services LLC, “Amazon Elastic Compute Cloud (Amazon EC2),” February 2012. [Online]. Accessed: 30 January 2012. Available: http://aws.amazon.com/ec2/

[3] Microsoft Corporation, “Microsoft Business Productivity Online Services and Software - BPOS,” February 2012. [Online]. Accessed: 30 January 2012. Available: http://www.microsoft.com/online/

[4] M. Cusumano, “Cloud computing and SaaS as new computing platforms,” *Communications of the ACM*, vol. 53, no. 4, p. 27, 2010. [Online]. Accessed: 30 January 2012. Available: http://portal.acm.org/citation.cfm?doid=1721654.1721667

[5] A. Meier and H. Stormer, *eBusiness & eCommerce*, E. Gosselin, Ed. Springer Publishing Company, Incorporated, 2005. [Online]. Accessed: 30 January 2012. Available: http://portal.acm.org/citation.cfm?id=1610392

[6] S. Harizopoulos, M. Shah, J. Meza, and P. Ranganathan, “Energy Efficiency: The New Holy Grail of Data Management Systems Research,” *Data Management*, vol. 80, no. 6, p. 4–7, 2009. [Online]. Accessed: 30 January 2012. Available: http://arxiv.org/abs/0909.1784

[7] Austrian Energy Agency, “Energy efficient servers in Europe,” Tech. Rep., October 2007. [Online]. Accessed: 30 January 2012. Available: http://www.efficient-datacenter.eu/fileadmin/docs/dam/downloads/public/Summary-Report.pdf

[8] T. Niemi and A.-P. Hameri, “Memory-based scheduling of scientific computing clusters,” *The Journal of Supercomputing*, pp. 1–25, 10.1007/s11227-011-0612-6. [Online]. Accessed: 30 January 2012. Available: http://dx.doi.org/10.1007/s11227-011-0612-6

[9] L. Barroso and U. Holzle, “The Case for Energy-Proportional Computing,” *Computer*, vol. 40, no. 12, pp. 33 –37, dec. 2007.

[10] V. Venkatachalam and M. Franz, “Power reduction techniques for

microprocessor systems," *ACM Comput. Surv.*, vol. 37, pp. 195–237, September 2005. [Online]. Accessed: 30 January 2012. Available: http://doi.acm.org/10.1145/1108956.1108957

[11] P. Padala, X. Zhu, Z. Wang, S. Singhal, K. G. Shin, P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin, "Performance evaluation of virtualization technologies for server consolidation," Tech. Rep., 2007.

[12] I. Tafa, E. Kajo, A. Bejleri, O. Shurdi, and A. Xhuvani, "The Performance between XEN-HVM , XEN-PV and Open-VZ during live-migration," *IJACSA International Journal of Advanced Computer Science and Applications*, vol. 2, no. 9, pp. 126–132, 2011. [Online]. Accessed: 30 January 2012. Available: www.ijacsa.thesai.org

[13] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 275–287, March 2007. [Online]. Accessed: 30 January 2012. Available: http://doi.acm.org/10.1145/1272998.1273025

[14] J. Wang, K.-L. Wright, and K. Gopalan, "XenLoop: a transparent high performance inter-vm network loopback," in *Proceedings of the 17th international symposium on High performance distributed computing*, ser. HPDC '08. New York, NY, USA: ACM, 2008, pp. 109–118. [Online]. Accessed: 30 January 2012. Available: http://doi.acm.org/10.1145/1383422.1383437

[15] Y. Hu, X. Long, J. Zhang, J. He, and L. Xia, "I/O scheduling model of virtual machine based on multi-core dynamic partitioning," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 142–154. [Online]. Accessed: 30 January 2012. Available: http://doi.acm.org/10.1145/1851476.1851494

[16] Oxford University Press, "Definition for virtualize - Oxford Dictionaries Online (World English)," December 2011. [Online]. Accessed: 30 January 2012. Available: http://oxforddictionaries.com/definition/virtualize

[17] D. King, E. Turban, J. K. Lee, J. McKay, and P. Marshall, *Electronic Commerce 2008*. Prentice Hall, 2007. [Online]. Accessed: 30 January 2012. Available: http://www.amazon.com/dp/0132243318

[18] S. Nanda and T. Chiueh, "A Survey on Virtualization Technologies," *Science*, vol. 179, no. Vm, pp. 1–42, 2005. [Online]. Accessed: 30 January 2012. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.74.371

[19] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. M. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith, "Intel virtualization technology," *Computer*, vol. 38, no. 5, pp. 48–56, 2005. [Online]. Accessed: 30 January 2012. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1430631

[20] M. Rosenblum and T. Garfinkel, "Virtual machine monitors: current technology and future trends," *Computer*, vol. 38, no. 5, pp. 39 – 47, may 2005.

[21] G. M. Amdahl, G. A. Blaauw, and F. P. Brooks, "Architecture of the IBM System/360," *IBM Journal of Research and Development*, vol. 8, no. 2, pp. 87 –101, April 1964.

[22] W. Vogels, "Beyond Server Consolidation," *Queue*, vol. 6, pp. 20–26, January 2008. [Online]. Accessed: 30 January 2012. Available: http://doi.acm.org/10.1145/1348583.1348590

[23] U. Steinberg and B. Kauer, "NOVA: a microhypervisor-based secure virtualization architecture," in *Proceedings of the 5th European conference on Computer systems*, ser. EuroSys '10. New York, NY, USA: ACM, 2010, pp. 209–222. [Online]. Accessed: 30 January 2012. Available: http://doi.acm.org/10.1145/1755913.1755935

[24] B. Gammage and P. Dawson, "Server Workloads : What Not to Virtualize Types of Virtualization," *System*, no. March 2008, 2010.

[25] F. Bellard, "QEMU, a fast and portable dynamic translator," 2005. [Online]. Accessed: 30 January 2012. Available: http://www.usenix.org/event/usenix05/tech/freenix/full_papers/bellard/bellard_html/

[26] L. Nussbaum, F. Anhalt, O. Mornard, and J.-P. Gelas, "Linux-based virtualization for HPC clusters," *Network*, no. Vm, pp. 221–234, 2009. [Online]. Accessed: 30 January 2012. Available: http://hal.inria.fr/inria-00425608/

[27] M. Casado and T. Koponen, "Virtualizing the Network Forwarding Plane," *Design*, pp. 8:1—8:6, 2010. [Online]. Accessed: 30 January 2012. Available: http://doi.acm.org/10.1145/1921151.1921162

[28] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three CPU schedulers in Xen," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, pp.

42–51, September 2007. [Online]. Accessed: 30 January 2012. Available: http://doi.acm.org/10.1145/1330555.1330556

[29] C. D. Patel, C. E. Bash, R. Sharma, M. Beitelmam, and R. J. Friedrich, *Smart cooling of data centers.* Google Patents, 2003.

[30] L. A. Barroso, "The price of performance," *Queue*, vol. 3, no. 7, p. 48, 2005. [Online]. Accessed: 30 January 2012. Available: http://portal.acm.org/citation.cfm?id=1095420

[31] Standard Performance Evaluation Corporation, "SPECpower_ssj2008 - ssj Design Document," February 2011. [Online]. Accessed: 30 January 2012. Available: www.spec.org/power/docs/SPECpower_ssj2008-Design_ssj.pdf

[32] L. Gray, A. Kumar, and H. Li, *Workload Characterization of the SPECpower_ssj2008 Benchmark.* Springer Berlin Heidelberg, 2008, vol. 5119, pp. 262–282. [Online]. Accessed: 30 January 2012. Available: http://www.springerlink.com/content/q5h13712v0512l55

[33] Standard Performance Evaluation Corporation, "IBM Corporation IBM System x3200 M2 SPECpower_ssj2008," February 2011. [Online]. Accessed: 30 January 2012. Available: http://www.spec.org/power_ssj2008/results/res2011q1/power_ssj2008-20110124-00340.html

[34] ——, "Fujitsu PRIMERGY TX300 S6 (Intel Xeon X5675) SPECpower_ssj2008," March 2011. [Online]. Accessed: 30 January 2012. Available: http://www.spec.org/power_ssj2008/results/res2011q1/power_ssj2008-20110222-00360.html

[35] R. P. Goldberg, "Architectural Principles for Virtual Computer Systems," Ph.D. dissertation, Harvard University, Cambridge, MA, 1972. [Online]. Accessed: 30 January 2012. Available: http://www.dtic.mil/cgi-bin/GetTRDoc?AD=AD772809&Location=U2&doc=GetTRDoc.pdf

[36] VMware, Inc., "Free VMware vSphere Hypervisor: Bare Metal Hypervisor (Based on VMware ESXi)," December 2011. [Online]. Accessed: 30 January 2012. Available: http://www.vmware.com/products/vsphere-hypervisor/

[37] Microsoft Corporation, "Microsoft Windows Server | Hyper-V | Virtualization | Virtual Server | HyperV Workload Performance," 2012. [Online]. Accessed: 30 January 2012. Available: http://www.microsoft.com/en-us/server-cloud/windows-server/hyper-v.aspx

[38] Oracle Corporation, *Oracle VM VirtualBox® User Manual*, December 2011. [Online]. Accessed: 30 January 2012. Available: http://download.virtualbox.org/virtualbox/4.1.8/UserManual.pdf

[39] J. Sugerman, G. Venkitachalam, and B.-H. Lim, *Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor.* USENIX Association, 2001, vol. 7, no. 2, pp. 1–14. [Online]. Accessed: 30 January 2012. Available: http://portal.acm.org/citation.cfm?id=715774

[40] A. Kivity, U. Lublin, and A. Liguori, "KVM: the Linux Virtual Machine Monitor," *Reading and Writing*, vol. 1, pp. 225–230, 2007. [Online]. Accessed: 30 January 2012. Available: http://www.kernel.org/doc/ols/2007/ols2007v1-pages-225-230.pdf

[41] 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," *Memory*, vol. 37, no. 5, pp. 164–177, 2003. [Online]. Accessed: 30 January 2012. Available: http://portal.acm.org/citation.cfm?id=945462

[42] N. FitzRoy-Dale, "The IA-32 processor architecture," Tech. Rep., May 2006. [Online]. Accessed: 30 January 2012. Available: http://www.eng.ucy.ac.cy/theocharides/courses/ece656/ia-32.pdf

[43] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 1: Basic Architecture*, December 2011. [Online]. Accessed: 30 January 2012. Available: http://www.intel.com/assets/pdf/manual/253665.pdf

[44] C. Pelletingeas, "Performance Evaluation of Virtualization with Cloud Computing," Ph.D. dissertation, IEEE. [Online]. Accessed: 30 January 2012. Available: http://researchrepository.napier.ac.uk/4010/

[45] XenSource, Inc., "3.2. Windows paravirtualized drivers," October 2007. [Online]. Accessed: 30 January 2012. Available: http://docs.vmd.citrix.com/XenServer/4.0.1/guest/ch03s02.html

[46] VMware, Inc., *Performance Best Practices for VMware vSphere 4.1*, October 2010. [Online]. Accessed: 30 January 2012. Available: http://www.vmware.com/pdf/Perf_Best_Practices_vSphere4.1.pdf

[47] The FreeBSD Project, *FreeBSD Manpages*, June 1993. [Online]. Accessed: 30 January 2012. Available: http://www.freebsd.org/cgi/man.cgi?query=chroot&sektion=2

[48] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer, “A secure environment for untrusted helper applications confining the Wily Hacker,” in *Proceedings of the 6th conference on USENIX Security Symposium, Focusing on Applications of Cryptography - Volume 6.* Berkeley, CA, USA: USENIX Association, 1996, pp. 1–1. [Online]. Accessed: 30 January 2012. Available: http://dl.acm.org/citation.cfm?id=1267569.1267570

[49] C. Greamo and A. Ghosh, “Sandboxing and Virtualization: Modern Tools for Combating Malware,” *IEEE Security Privacy Magazine*, vol. 9, no. 2, pp. 2011–2011, 2011. [Online]. Accessed: 30 January 2012. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5739643

[50] J. Ahola, A. Ashraf, B. Gupta, U. Hassan, M. Hartikainen, M. Helenius, K. Heljanko, J. Järvenpää, A. Kallio, J. Kannisto, and et al., “Best Practices for Cloud Computing,” Tech. Rep., 2010. [Online]. Accessed: 30 January 2012. Available: http://www.cloudsoftwareprogram.org/rs/2234/9ed65124-0873-400e-bc8a-9c85c1f1afa8/63b/filename/d1-1-2-techreportbestpracticesforcloudcomputing.pdf

[51] PayPal, Inc., *PayPal Sandbox User Guide*, October 2007. [Online]. Accessed: 30 January 2012. Available: https://www.paypalobjects.com/de_DE/pdf/PP_Sandbox_UserGuideDE.pdf

[52] B. Hayes, “Cloud Computing: As Software Migrates from Local PCs to Distant Internet Servers, Users and Developers Alike Go Along for the Ride,” *Communications of the ACM*, vol. 51, no. 2, pp. 9–11, 2008.

[53] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, “A break in the clouds: towards a cloud definition,” *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 50–55, December 2008. [Online]. Accessed: 30 January 2012. Available: http://doi.acm.org/10.1145/1496091.1496100

[54] Q. Zhang, L. Cheng, and R. Boutaba, “Cloud computing: state-of-the-art and research challenges,” *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010. [Online]. Accessed: 30 January 2012. Available: http://www.springerlink.com/index/10.1007/s13174-010-0007-6

[55] B. Armbrust, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin, “A view of cloud computing,” *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010. [Online]. Accessed: 30 January 2012. Available: http://portal.acm.org/citation.cfm?id=1721672

[56] E. Ciurana, *Developing with Google App Engine.* Springer, 2009. [Online]. Accessed: 30 January 2012. Available: http://books.google.com/books?hl=en&lr=&id=_ks9HQLVxaAC&pgis=1

[57] R. Attebury, J. George, C. Judd, B. Marcum, and N. Montgomery, "Google docs: a review," *Against the Grain*, vol. 20, no. 2, pp. 38,40,42, 2008.

[58] Amazon.com, Inc., "Amazon Elastic Compute Cloud (Amazon EC2)," 2008. [Online]. Accessed: 30 January 2012. Available: http://aws.amazon.com/ec2/

[59] Zenoss, Inc., "2010 Virtualization and Cloud Computing Survey," 2010. [Online]. Accessed: 30 January 2012. Available: http://www.zenoss.com/in/virtualization_survey.html

[60] E. A. Lee, "The Problem with Threads," *Computer*, vol. 39, no. 5, pp. 33–42, 2006. [Online]. Accessed: 30 January 2012. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1631937

[61] S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, R. L. Stamm, and D. M. Tullsen, "Simultaneous Multithreading: A Platform for Next-Generation Processors," *IEEE Micro*, vol. 17, no. 5, pp. 12–18, 1997. [Online]. Accessed: 30 January 2012. Available: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=621209

[62] T. L. Martin and D. P. Siewiorek, "Nonideal battery and main memory effects on CPU speed-setting for low power," *IEEE Trans Very Large Scale Integr Syst*, vol. 9, no. 1, pp. 29–34, 2001. [Online]. Accessed: 30 January 2012. Available: http://portal.acm.org/citation.cfm?id=375817.375831

[63] Intel Corporation, "Enhanced Intel SpeedStep® Technology and Demand-Based Switching on Linux," 2009. [Online]. Accessed: 30 January 2012. Available: http://software.intel.com/en-us/articles/enhanced-intel-speedstepr-technology-and-demand-based-switching-on-linux/

[64] Advanced Micro Devices, Inc., "Energy-efficient AMD Desktop Processors," January 2008. [Online]. Accessed: 30 January 2012. Available: http://www.amd.com/us/Documents/43029A_Brochure_PFD.pdf

[65] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation, *Advanced Configuration and Power Interface Specification*, December 2011. [Online]. Accessed: 30 January 2012. Available: http://www.acpi.info/spec.htm

[66] Intel Corporation, "Intel® Turbo Boost Technology in Intel® Core™ Microarchitecture (Nehalem) Based Processors," Tech. Rep., November 2008. [Online]. Accessed: 30 January 2012. Available: http://download.intel.com/design/processor/applnots/320354.pdf

[67] D. Koufaty and D. T. Marr, "Hyperthreading technology in the netburst microarchitecture," *IEEE Micro*, vol. 23, no. 2, pp. 56–65, 2003. [Online]. Accessed: 30 January 2012. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1196115

[68] Intel Corporation, "First the Tick, Now the Tock: Intel® Microarchitecture (Nehalem)," Tech. Rep., 2009. [Online]. Accessed: 30 January 2012. Available: http://www.intel.com/technology/architecture-silicon/next-gen/319724.pdf

[69] J. R. Bulpin and I. A. Pratt, "Hyper-threading aware process scheduling heuristics," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ser. ATEC '05. Berkeley, CA, USA: USENIX Association, 2005, pp. 27–27. [Online]. Accessed: 30 January 2012. Available: http://dl.acm.org/citation.cfm?id=1247360.1247387

[70] Y. Ruan, V. S. Pai, E. Nahum, and J. M. Tracey, "Evaluating the Impact of Simultaneous Multithreading on Network Servers Using Real Hardware," *SIGMETRICS 05 Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, vol. 33, no. 1, p. 315, 2005. [Online]. Accessed: 30 January 2012. Available: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed&cmd=Retrieve&dopt=AbstractPlus&list_uids=1064212.1064254

[71] R. Russell, "virtio: towards a de-facto standard for virtual I/O devices," *SIGOPS Oper. Syst. Rev.*, vol. 42, pp. 95–103, July 2008. [Online]. Accessed: 30 January 2012. Available: http://doi.acm.org/10.1145/1400097.1400108

[72] D. G. Murray, G. Milos, and S. Hand, "Improving Xen security through disaggregation," *Proceedings of the fourth ACM SIGPLANSIGOPS international conference on Virtual execution environments VEE 08*, pp. 151–160, 2008. [Online]. Accessed: 30 January 2012. Available: http://portal.acm.org/citation.cfm?id=1346256.1346278

[73] S. Garfinkel and G. Spafford, *Practical UNIX & Internet Security.* O'Reilly & Associates, 1996, no. April. [Online]. Accessed: 30 January 2012. Available: http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/0596003234

[74] "lxc Linux Containers," December 2011. [Online]. Accessed: 30 January 2012. Available: http://lxc.sourceforge.net/

[75] M. Bardac, R. Deaconescu, and A. M. Florea, "Scaling Peer-to-Peer testing using Linux Containers," in *Roedunet International Conference (RoEduNet), 2010 9th*, june 2010, pp. 287 –292.

[76] P. Menage, *CGROUPS*, August 2011. [Online]. Accessed: 30 January 2012. Available: http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

[77] Electronic Educational Devices, *Watts up? Manual 7.0 b (Page 2 - 3)*, February 2008. [Online]. Accessed: 30 January 2012. Available: https://www.wattsupmeters.com/secure/downloads/manual_rev_9_corded0812.pdf

[78] D. L. Streiner, "Maintaining standards: differences between the standard deviation and standard error, and when to use each," *Canadian Journal of Psychiatry*, vol. 41, no. 0706-7437 (Print) LA - eng PT - Journal Article SB - IM, pp. 498–502, 1996.

[79] J. Wei, X. Zhang, G. Ammons, V. Bala, and P. Ning, "Managing security of virtual machine images in a cloud environment," *Proceedings of the 2009 ACM workshop on Cloud computing security CCSW 09*, vol. 28, no. Vm, p. 91, 2009. [Online]. Accessed: 30 January 2012. Available: http://portal.acm.org/citation.cfm?doid=1655008.1655021

[80] D. Brodowski, *CPU frequency and voltage scaling code in the Linux(TM) kernel*, July 2011. [Online]. Accessed: 30 January 2012. Available: http://www.kernel.org/doc/Documentation/cpu-freq/governors.txt

[81] R. Longbottom, "Roy Longbottom's PC Benchmark Collection," November 2011. [Online]. Accessed: 30 January 2012. Available: http://www.roylongbottom.org.uk

[82] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK Benchmark: past, present and future," *Concurrency and Computation Practice and Experience*, vol. 15, no. 9, pp. 803–820, 2003. [Online]. Accessed: 30 January 2012. Available: http://doi.wiley.com/10.1002/cpe.728

[83] B. Martin, "Using Bonnie++ for filesystem performance benchmarking," 2008. [Online]. Accessed: 30 January 2012. Available: http://archive09.linux.com/feature/139742

[84] M. Egli and D. Gugelmann, *Iperf - network stress tool*, 2007. [Online]. Accessed: 30 January 2012. Available: http://wo-ist.net/files/iperf.pdf

[85] D. Mosberger and T. Jin, “Httperf, a tool for measuring web server performance,” *First Workshop on Internet Server Performance*, vol. 26, pp. 31–37, 1998.

[86] United States Environmental Protection Agency, “Report to Congress on Server and Data Center Energy Efficiency Public Law 109-431,” *Environmental Protection*, vol. 109, p. 431, 2007. [Online]. Accessed: 30 January 2012. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.136.3058&rep=rep1&type=pdf

[87] Red Hat, Inc., “KVM – Kernel-based Virtual Machine,” Tech. Rep., September 2008. [Online]. Accessed: 30 January 2012. Available: http://www.redhat.com/resourcelibrary/whitepapers/doc-kvm

[88] Xen.org, *Xen 4.1 Data Sheet*, March 2011. [Online]. Accessed: 30 January 2012. Available: http://xen.org/files/Xen_4_1_Datasheet.pdf

[89] VMware, Inc., “Host Power Management in VMware vSphere 5,” Tech. Rep., August 2011. [Online]. Accessed: 30 January 2012. Available: http://www.vmware.com/files/pdf/hpm-perf-vsphere5.pdf

[90] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehörster, and A. Brinkmann, *Non-intrusive Virtualization Management using libvirt*, 2010, p. 574–579. [Online]. Accessed: 30 January 2012. Available: http://portal.acm.org/citation.cfm?id=1871061

APPENDIX 1: OUTPUT SAMPLES

Watts up? PRO log

```
2011-10-18 18:25:05;1318955105;118.9;0.0
2011-10-18 18:25:06;1318955106;144.6;0.0
2011-10-18 18:25:07;1318955107;134.9;0.1
2011-10-18 18:25:08;1318955108;187.2;0.1
2011-10-18 18:25:09;1318955109;194.3;0.2
2011-10-18 18:25:10;1318955110;194.1;0.2
2011-10-18 18:25:11;1318955111;193.9;0.3
2011-10-18 18:25:12;1318955112;193.2;0.3
2011-10-18 18:25:13;1318955113;145.1;0.4
2011-10-18 18:25:14;1318955114;136.6;0.4
2011-10-18 18:25:15;1318955115;123.2;0.4
2011-10-18 18:25:16;1318955116;123.2;0.5
2011-10-18 18:25:17;1318955117;116.1;0.5
2011-10-18 18:25:18;1318955118;134.4;0.5
2011-10-18 18:25:19;1318955119;121.2;0.6
2011-10-18 18:25:20;1318955120;121.1;0.6
2011-10-18 18:25:21;1318955121;98.4;0.6
2011-10-18 18:25:22;1318955122;90.1;0.7
2011-10-18 18:25:23;1318955123;85.0;0.7
```

Linpack

```
 ##################################################

   Assembler CPUID and RDTSC
  CPU GenuineIntel, Features Code BFEBFBFF, Model Code 000106A5
   Intel(R) Xeon(R) CPU           E5520  @ 2.27GHz
   Measured - Minimum 2261 MHz, Maximum 2261 MHz
   Linux Functions
   get_nprocs() - CPUs 8, Configured CPUs 8
   get_phys_pages() and size - RAM Size  7.07 GB, Page Size 4096 Bytes
   uname() - Linux, milka, 3.0.0
   #3 SMP Tue Aug 23 15:13:06 CEST 2011, x86_64

 ##################################################

  Linpack Double Precision Unrolled Benchmark n @ 100
  Optimisation Opt 3 64 Bit, Thu Sep 29 11:27:50 2011

  Speed     1879.49 MFLOPS

  Numeric results were as expected
```

iperf

```
------------------------------------------------------------
Client connecting to 192.168.2.235, TCP port 5001
TCP window size: 16.0 KByte (default)
------------------------------------------------------------
[  4] local 192.168.2.232 port 43710 connected with 192.168.2.235 port 5001
[  3] local 192.168.2.232 port 43709 connected with 192.168.2.235 port 5001
[  5] local 192.168.2.232 port 43711 connected with 192.168.2.235 port 5001
[  6] local 192.168.2.232 port 43712 connected with 192.168.2.235 port 5001
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-600.0 sec  16.7 GBytes   239 Mbits/sec
[  3]  0.0-600.0 sec  16.1 GBytes   231 Mbits/sec
[  6]  0.0-600.1 sec  16.1 GBytes   231 Mbits/sec
[  5]  0.0-600.1 sec  16.8 GBytes   240 Mbits/sec
[SUM]  0.0-600.1 sec  65.8 GBytes   942 Mbits/sec
```

httperf

```
1318955105.014323 0.737685
1318955105.112467 0.735838
...
1318958446.873909 0.000559
1318958446.977073 0.003730
Maximum connect burst length: 1

Total: connections 33434 requests 33434 replies 33424 test-duration 3343.301 s

Connection rate: 10.0 conn/s (100.0 ms/conn, <=83 concurrent connections)
Connection time [ms]: min 0.3 avg 59.6 max 13380.1 median 0.5 stddev 399.5
Connection time [ms]: connect 0.2
Connection length [replies/conn]: 1.000

Request rate: 10.0 req/s (100.0 ms/req)
Request size [B]: 96.0

Reply rate [replies/s]: min 1.0 avg 10.0 max 17.0 stddev 0.6 (668 samples)
Reply time [ms]: response 30.1 transfer 29.3
Reply size [B]: header 217.0 content 5057.0 footer 0.0 (total 5274.0)
Reply status: 1xx=0 2xx=31781 3xx=2 4xx=1641 5xx=0

CPU time [s]: user 799.43 system 2542.73 (user 23.9% system 76.1% total 100.0%)
Net I/O: 52.4 KB/s (0.4*10^6 bps)

Errors: total 10 client-timo 10 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
```

OpenMP

```
##################################################

  Assembler CPUID and RDTSC
 CPU GenuineIntel, Features Code BFEBFBFF, Model Code 000106A5
  Intel(R) Xeon(R) CPU           E5520  @ 2.27GHz
  Measured - Minimum 2261 MHz, Maximum 2261 MHz
  Linux Functions
  get_nprocs() - CPUs 8, Configured CPUs 8
  get_phys_pages() and size - RAM Size  7.82 GB, Page Size 4096 Bytes
  uname() - Linux, milka, 3.0.0
  #3 SMP Tue Aug 23 15:13:06 CEST 2011, x86_64

##################################################

  64 Bit OpenMP MFLOPS Benchmark 1 Mon Oct 24 16:19:45 2011

                  Via Ubuntu 64 Bit Compiler

  Test               4 Byte  Ops/   Repeat    Seconds   MFLOPS       First   All
                      Words  Word   Passes                         Results  Same

 Data in & out       100000     2     2500   0.084233     5936    0.929538   Yes
 Data in & out      1000000     2      250   0.078038     6407    0.992550   Yes
 Data in & out     10000000     2       25   0.172303     2902    0.999250   Yes

 Data in & out       100000     8     2500   0.183409    10905    0.957117   Yes
 Data in & out      1000000     8      250   0.178034    11234    0.995517   Yes
 Data in & out     10000000     8       25   0.192494    10390    0.999549   Yes

 Data in & out       100000    32     2500   0.584376    13690    0.890211   Yes
 Data in & out      1000000    32      250   0.578125    13838    0.988082   Yes
 Data in & out     10000000    32       25   0.582955    13723    0.998796   Yes
```

Bonnie++

```
Started: Tue Jan 24 15:49:59 CET 2012
Version  1.96       ------Sequential Output------ --Sequential Input- --Random-
Concurrency   1     -Per Chr- --Block-- -Rewrite- -Per Chr- --Block-- --Seeks--
Machine        Size K/sec %CP K/sec %CP K/sec %CP K/sec %CP K/sec %CP  /sec %CP
milka            4G   881  97 104951   7 37805   5  3823  94 124686  10 233.5   5
Latency              9274us     603ms    1908ms    9699us   25248us    2139ms
Version  1.96       ------Sequential Create------ --------Random Create--------
milka               -Create-- --Read--- -Delete-- -Create-- --Read--- -Delete--
              files  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP  /sec %CP
                400 49167  61 725102 99  1095   1 51502  61 +++++ +++  1135   1
Latency               653ms     442us   17804ms     641ms      12us   16109ms
1.96,1.96,milka,1,1327415729,4G,,881,97,104951,7,37805,5,3823,94,124686,10,233.5,
5,400,,,,,49167,61,725102,99,1095,1,51502,61,+++++,+++,1135,1,9274us,603ms,1908ms,
9699us,25248us,2139ms,653ms,442us,17804ms,641ms,12us,16109ms
Finished: Tue Jan 24 16:06:19 CET 2012
```