UNIVERSITY OF AMSTERDAM
Faculty of Physics, Mathematics and Informatics
MSc System and Network Engineering
Final Research Project

Power Efficiency of Hypervisor-based Virtualization versus Container-based Virtualization

February 11, 2016

Author:
Jeroen van Kessel

Supervisors:
dr. ir. Arie Taal
dr. Paola Grosso

Abstract

Up until recently, hypervisor-based virtualization platforms dominated the virtualization industry. However, container-based virtualization, an alternative to hypervisor-based virtualization, simplifies and fastens the deployment of virtual entities. Relevant research has already shown that container-based virtualization either performs equally or better than hypervisor-based virtualization in terms of performance in almost all cases. This research project investigates whether the power efficiency significantly differs on Xen, which is based on hypervisor virtualization, and Docker, which is based on container-based virtualization. The power efficiency is obtained by running synthetic applications and measuring the power usage on different hardware components. Rather than measuring the overall power of the system, or looking at empirical studies, hardware components such as CPU, memory and HDD will be measured internally by placing power sensors between the motherboard and circuits of each measured hardware component. This newly refined approach shows that both virtualization platforms behave roughly similar in IDLE state, when loading the memory and when performing sequential writes for the HDD. Contrarily, the results of CPU and sequential HDD reads show differences between the two virtualization platforms, where the performance of Xen is significantly weaker in terms of power efficiency.

Keywords: Linux Containers, Hypervisor Virtualization, Power Efficiency

Contents

1 Introduction

Throughout the development of data centers, many researchers have investigated the power consumption of servers with respect to a certain factor such as performance [1] or power efficiency [2]. This new research will attempt to evaluate the power efficiency of hardware component, the CPU, Memory, and HDD, when using a traditional hypervisor versus a container-based virtualization approach.

Virtualization is extensively implemented in modern data centers. According to Cisco, by the year of 2019, more than 86 percent of all workload will be processed by cloud data centers, while 14 percent will be processed by other data centers [3]. Most of these running virtual nodes are handled by a hypervisor such as KVM [4], Xen [5] or vSphere [6]. These hypervisors, however, come with certain overhead when deploying virtual nodes, since isolation and resource control is desired. This virtualization method leads to the deployment of separate operating systems for each virtual node. Besides the computational overhead, migrating virtual nodes is still challenging due to compatibility and integration issues.

Container-based virtualization is a different approach of deploying virtual nodes based on a shared operating system kernel. Rather than deploying a full operating system, container-based virtualization modifies an existing operating system to provide extra isolation and proves to be more resource efficient [7]. An application with all of its dependencies is deployed into a container which can then be moved to any infrastructure platform independent of its system architecture. Docker [8] is such an open-source implementation of this container-based virtualization technology.

Besides performance gain, power efficiency is a factor which should be taken into account. According to measurements from the Earth System Research Laboratory, the amount of carbon dioxide in the Earth's atmosphere has risen above 400 parts per million (ppm) for the first time in 3 million years [9]. This means that the global CO_2 levels have permanently exceeded the safety zone. The extensive use of cloud services also contribute to the yearly rise of carbon dioxide CO_2 emissions. Furthermore, energy consumption is a prominent cost factor for any data center. Therefore, in the United States, the Environmental Protection Agency (EPA) has proposed that large data centers use energy meters as a first step toward creating operating-efficiency standards [10]. The European Union has issued a voluntary code of conduct laying out best practises for running data centers at higher levels of energy efficiency [11].

The research presented in this thesis will attempt to quantify and compare the power consumption of the overhead of hypervisor versus container-based virtualization. This research will be part of a larger project called 'Greening the Cloud' [12], which aims to develop a larger framework in order to label the 'greenness' of data centers by researching the impact of software on hardware [13].

The outline of this thesis is as follows. Chapter 2 will discuss the previous academic work. Next, Chapter 3 discusses the research questions of this project. Chapter 4 briefly introduces the concepts of both virtualization platforms. Next, Chapter 5 describes the experimental setup and Chapter 6 elaborates on the approach and the calculation methodology. Chapter 7 presents and elaborates on the acquired results of this research. Lastly, Chapters 8 and 9 elaborate on the conclusions of this research together with future directions.

2 Related Work

Both power and performance efficiency of hypervisors and Linux containers have been researched in the past. Power models for virtual nodes exist but the coefficient of each component is often obtained through empirical studies [14]. This research will measure each prominent hardware component independently instead of using a power measurement device. The following studies are relevant to this project.

C. van der Poll (2015) studied the resource usage in hypervisors [15]. This research presents the comparison of power consumption of two open-source hypervisors, KVM and Xen. Van der Poll uses the stress utility [16] to impose load on the CPU, memory and HDD. The results of these tests point towards KVM as a more green solution than Xen. However, container-based virtualization was not mentioned during this research. This new research builds onto the conclusion and future work of van der Poll. The author proposed a quantitative power consumption and performance scale model of the server's CPU, Memory and HDD I/O. Furthermore, van der Poll used several benchmark scripts to specify the duration and to create a well defined result overview. Therefore, van der Poll's scripts will be customised and re-used during this research project.

X. Peng and Z. Sai (2013) developed a power model for virtual nodes [17]. This power model measures the virtual node's power consumption running on an Xen hypervisor. X. Peng and Z. Sai took the following components into account: processor, memory, Hard Disk Drive and I/O controller. Their measuring technique is claimed to be low-cost and effective for virtualized resources in large-scale data centers. Therefore, this power model will be studied although container-based virtualization might not be applicable to this power model. This research concluded that this power model is effective at improving the power efficiency when the virtualization ratio of a data center is high.

In 2014, the IBM research division made a performance comparison of virtual nodes ran on a hypervisor versus Linux containers, where KVM was used as a hypervisor and Docker as container-based virtualization software [7]. The IBM researchers demonstrated that Docker had equal or faster performance compared to KVM. The researchers used the following synthetic applications to stress the hardware components: LINPACK [18] for the CPU, STREAM [21] for memory and fio [22] for stressing the HDD. LINPACK benchmarks the CPU performance of a server using a linear algebra problem. Figure 1 on the next page shows a specific use case from this research with 45000 linear algebra problems. Although the performance of KVM and Docker was thoroughly researched, power efficiency was not taken into account.

Like the IBM research division, the Ericsson research division (2015) also made performance comparison between KVM and Xen, which are based on hypervisor virtualization, and Docker and LXC, which are based on container virtualization [1]. This study also uses LINPACK to stress the CPU, STREAM for the memory and Bonnie++ [25] to benchmark the HDD. The authors concluded that the relative differences in performance between the different platforms are not substantial. Furthermore, the performance results from this research will be compared. Little difference in terms of performance results should be measured such that the power efficiency of the different virtualization platforms can be determined later on.

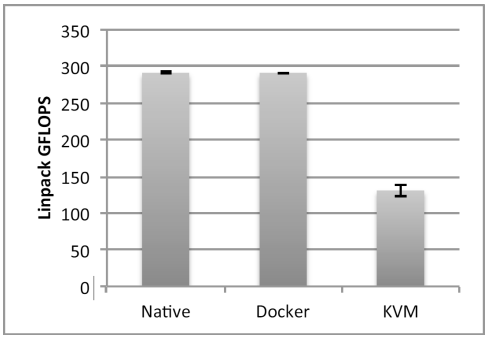


Figure 1: LINPACK performance on two physical CPUs, all cores dedicated to the virtual entities. Mean obtained from ten runs. Error bars indicate the standard deviation obtained over all runs. [7]

In December 2015, the Ericsson researcher released another empirical study [2]. This time, power consumption of KVM, Xen, Docker and LXC was researched. This study uses sysbench [24] to stress the CPU and memory, while the HDD was not taken into account. The result of this study shows, despite the number of virtual nodes running, Xen and Docker behave similarly in IDLE state and in CPU/ Memory stress test. However, this research does not measure each hardware component individually, but instead uses a power measurement device. Therefore, the author reports the average power consumed by the platforms when increasing the number of virtual nodes and cores. This thesis will disentangle the various components and provides the power efficiency metric (performance per watt). Power consumed alone is not as informative as the efficiency, which will be investigated in this new research. Nonetheless, results of this second Ericsson research will be compared. This research will use LINPACK for the CPU, sysbench for memory and Bonnie++ for stressing the HDD.

3 Research Questions

This research aims to quantify and compare the power consumption of the overhead of a traditional hypervisor-based virtualization versus container-based virtualization. The primary research question of this research is as follows:

Is there a measurable difference in power efficiency when running an application under a traditional hypervisor-based virtualization versus Linux containers?

Answering this research question will allow to analyse whether the power efficiency is different per hardware component while running a virtual node under a hypervisor versus a Linux container-based application.

To answer this research question, one must thoroughly define a methodology and install power sensors between the motherboard and circuits of each hardware component. Therefore, the following sub-question needs to be answered:

How does one measure the power consumption of each hardware component internally in the most accurate way?

The next chapter explains the different virtualization platforms that will be used during the course of this research.

4 Theoretical Framework

Currently, there are two widely used virtualization technologies, namely hypervisor-based virtualization and container-based virtualization [26]. This chapter describes these two virtualization concepts in more detail.

4.1 Traditional Hypervisor-based Virtualization

Virtualization is, at its foundation, 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 [27]. This includes making a single physical resource, such as an operating system or application, appear to function as multiple logical resources. Therefore, virtualization makes multiple physical machines, such as server, appear as a single logical resource [27]. In other words, the traditional concept of virtualization is based on emulating virtual hardware by using a hypervisor, which can be inefficient in terms of system resources.

Unlike containers, which are based on a shared kernel, hypervisors deploy a separate kernel for each virtual node, also known as a virtual machine [28]. Nonetheless, hypervisors allow the deployment of operating systems that are heterogeneous to the host machine. This means a hypervisor is capable of deploying operating systems with different kernel versions simultaneously. This is illustrated in Figure 2. The next subsection describes the specification of one of these hypervisors, namely Xen. Xen will be used in this research project as a traditional hypervisor.

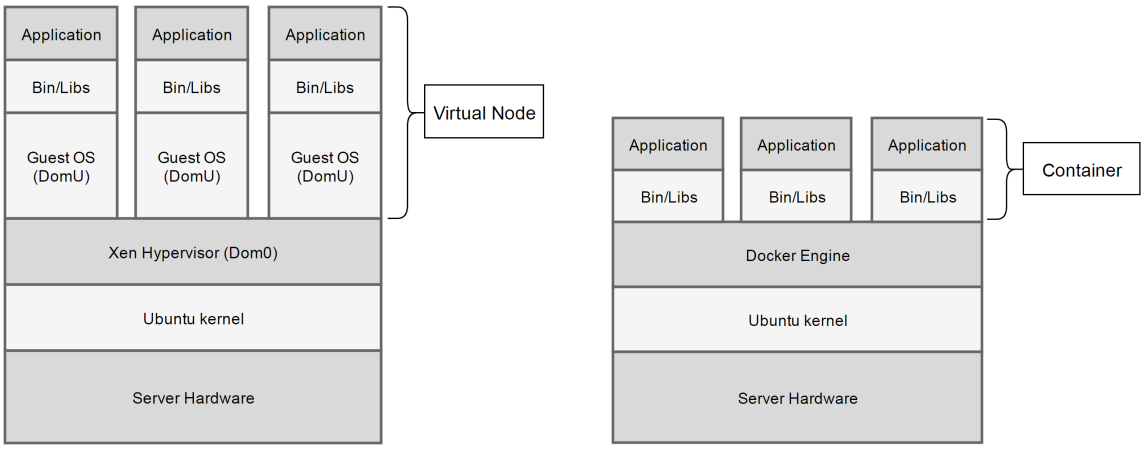


Figure 2:
(2a): Hypervisor-based Virtualization Architecture
(2b): Container-based Virtualization Architecture.
Figure is based on the diagram in [29]

4.1.1 Xen Hypervisor

Xen is an open-source bare-matel hypervisor. This makes it possible to run a finite number of virtual nodes in parallel on a single host [5]. Cloud providers such as Amazon Web Services and Rackspace Public Cloud use Xen hypervisor as its foundation. Xen is also integrated in OpenStack orchestration software. This research will use the 64-bit Ubuntu 15.10 `4.2.0-19-generic` kernel, which supports direct kernel integration with Xen 4.5.1 hypervisor. Xen offers

different modes of deploying a virtual node, namely paravirtualization (PV), hardware virtual machine (HVM) and a combined (PVHVM) mode, which are explained in the next sections.

4.1.2 Xen Paravirtualization

Paravirtualization (PV) is a software virtualization mode that allows for the deployment of virtual nodes. The virtual nodes must have a modified kernel. Xen paravirtualized nodes are aware of the hypervisor, and do not have any virtual emulated hardware [30]. Therefore, Paravirtualized nodes make direct calls without an additional layer [30]. Paravirtualization generally has less overhead and is faster at deploying virtual nodes compared to HVM. However, system calls inside the virtual node need to be translated and sent to the host machine's kernel. This extra step reduces performance. Intel processors use VT-x to fix this problem by providing virtualization hardware instructions and eliminating the system call translation step [41].

4.1.3 Xen HVM Virtualization

Hardware Virtual Machine (HVM), also known as Hardware Assisted Virtualization, is a processor virtualization mode that requires the extension from the host processor. Intel processors complies to this extension by using hardware virtualization, which is addressed as VT-x. This processor extension uses the privileged instructions, syscalls and the page tables. This results in better performance compared to paravirtualization [30]. Furthermore, HVM nodes do not require a modified kernel for virtual nodes. Therefore, heterogeneous kernels other than the host kernel can be deployed simultaneously as virtual nodes.

HVM uses the QEMU (Quick EMUlator) hardware emulator. The HVM QEMU emulates the processor through dynamic binary translation and allocates them as vCPUs. In order to optimise the use of synthetic benchmark programs and to separate the CPU workload of the virtual node from the hypervisor, vCPU topology `vcpu-pin` will be configured for the research presented to exclusively pin physical CPU cores to the vCPUs of the virtual node [31]. This should prevent the virtual node from swapping to a vCPU cores used by hypervisor (scheduler) [32].

During the course of this research project, HVM proved to use the host processor topology for better vCPU allocation and performance. This is because vCPU topology is not yet supported on paravirtualization virtual nodes [33]. Therefore, paravirtualization was not reseached during this research.

4.1.4 Xen PVHVM Virtualization

The PVHVM modus is a combination of PV and HVM, where HVM is used as the underlying mode in combination with paravirtual drivers. These paravirtual drivers are included in the Ubuntu 15.10 `4.2.0-19-generic` kernel which bypass the emulation for HDD and the Network Interface Card (NIC). These paravirtual drivers result in better performance of HVM virtual nodes [30]. Another mode is PVH, which is currently considered experimental. Therefore, PVH mode will not be used nor studied during this research project [30].

In this research project, a PVHVM virtual node will be deployed in order to make a fair comparison between an optional Xen Hypervisor environment and a container-based virtualization platform on Docker. The configuration file for this virtual node is attached in Appendix A. Figure 2a illustrates the high-level architecture of Xen on top of the Ubuntu kernel.

4.2 Linux Container-based Virtualization

Linux container-based virtualization provides a different level of abstraction without a hypervisor. Container-based virtualization results in a lighter virtual environment compared to hypervisor-based virtualization [34], because containers do not need a separate kernel. Instead, all hardware will be simulated [35]. Therefore, Linux containers are much more efficient and smaller in terms of compute resources compared to hypervisor-based virtual nodes. While hypervisors virtualize hardware, containers are deployed on top of the Linux kernel.

Container-based virtualization deploys a 'container', which is an execution environment that shares the host kernel of the host system and is isolated from other containers in the system [39]. A container thus consists of an operating system, user-added files, and meta-data [28] as shown in the previous Figure 2b. Because containers result in a significant decrease in compute resources needed to run an actual application, the deployment density can be increased compared to a hypervisor-based virtualization platform.

Most container-based virtualization platforms use the LXC (LinuX Container) toolset, which takes care of sandboxing processes from one another, such that each container has its own vCPU, memory and file system [35]. This LXC toolset uses the same principle to abstract the operating system kernel [34]. Many container-based virtualization platforms exist, such as OpenVZ [36], VServer [37], Google's container platform lmctfy [38] - 'Let Me Contain That For You' and, of course, Docker [8]. Docker, which is the industry standard, is used in this research project and described in more detail in the next section.

4.2.1 Docker Engine

Docker is an open-source platform which can build, ship, and run applications without the deployment of a complete guest operating system [8]. Therefore, container-based virtualization allows the deployment of applications with less overhead. Docker packages an application, including its libraries and dependencies, into a 'container'. Docker used to rely on the LXC toolset, but recently released its own libraries called `libcontainer` [39]. `libcontainer` deploys containers with namespaces and control groups [39]. These Linux kernel features are described in the next section.

4.2.2 Docker Namespaces

Namespaces have existed for many years in Unix world. Namespaces are used to limit the scope of kernel-side names and data structures [40]. In 1998, FreeBSD could already isolate processes using jails to improve security [41]. Namespaces in Docker are also used to isolate containers, such that they have their own view of the system [28]. In basic terminology, namespaces limit what you can see, and therefore what a container can use. Namespaces are created with the `clone()` system call [40]. Namespaces provide a layer of isolation. Each aspect of a container runs in its own namespace and does not have access outside it. Docker uses namespaces for process IDs, network devices, mount points and kernel isolation [40].

4.2.3 Docker Control Groups

Control groups, also addressed as cgroups, take care of the resource limitation of the hardware component, such as vCPU, memory and disk I/O [40]. This research project uses cgroups to limit the memory and vCPU allocation to containers.

4.2.4 Docker Images

Docker uses images as a foundation to deploy containers. Docker has its own centralised repository for available images called 'Docker Hub' [42]. This Docker Hub platform allows users to download and update images, such as Red Hat, Ubuntu, Debian, openSUSE and CentOS [42]. Also, applications such as NGINX, MongoDB and Tomcat deployed on top of a Linux operating system, can be pulled and started in real-time [43]. These pre-compiled Docker images may use different Linux kernels, but the host operating system kernel is shared when possible. It should be noted that Docker cannot deploy heterogeneous operating systems such as Windows. The commands for deploying the container is attached in Appendix B.

4.2.5 Union File Systems

Docker images use the Union file systems [44], which separate file systems by known branching. UnionFS uses layers, forming a single file system [28]. Rather than replacing the whole image, only the specific layer is added or updated [28]. When Docker deploys a container from an image, it first mounts the root file system as read only. However, instead of making the file system read-write, another file system layer is attached to the container. This process continues every time a change occurs to the file system of the container [41]. This whole process results in a hierarchical tree structure of the container's file system, which is visualised in Table 1 using the `dockviz` Docker utility [45].

```
dockviz images -t -i
|--511136ea335a Virtual Size: 0.0 B
  |--f10ebce230e1 Virtual Size: 101.7 MB
  | |-82cdea73b5b5 Virtual Size: 255.5 KB
  |  |--5dbd93b5a02f Virtual Size: 1.9 KB
  |    |--74338d11401 Virtual Size: 105.7 MB Tags: ubuntu:15.04
  |--ef519c93291a Virtual Size: 105.6 MB
  | |--07302303becc Virtual Size: 251.0 KB
  |   |--cf83c907452c Virtual Size: 1.9 KB
  |     |--a3cf8ae4e998 Virtual Size: 70.1 MB Tags: ubuntu:15.10
```

Table 1: `dockviz` incremental file system layers. Two independent Ubuntu 15.04 and Ubuntu 15.10 containers where each file system change is branched [45]

5 Experimental Setup

Rather than looking at empirical studies or measuring the total power consumption of the servers' Power Supply Unit (PSU), specific hardware components are measured independently. This section describes the experimental setup and how the power consumption data is obtained together with the calculation methodology.

5.1 Greening the Cloud and SEFlab

Today's data centers consume enormous amounts of energy, most of which is generated from environmentally-unfriendly fossil fuels [46]. The Greening the Cloud project investigates the energy efficient of virtualized efficiency without sacrificing performance. Therefore, investigating utilisation with respect to power efficiency may lead to unnecessary waste of energy in data centers. This research projects contributes to this larger framework by investigating the power efficiency of industry standard virtualization platforms.

In order to answer the research questions of this project, one must study the power consumption and efficiency of each hardware component independently. The Software Energy Footprint laboratory (SEFlab) [13] of the HvA provides measurement equipment and servers to conduct these measurements. Therefore, the SEFlab facilitated this research project. Furthermore, the Greenlab of the VU [47] also offers a server for research. This server is maintained and owned by the SEFlab.

5.2 Power Consumption Sensors

In order to calculate the power dissipation of each hardware component, the current and voltage needs to be measured internally. This is done by installing so called power sensors between the motherboard and circuits of each hardware component as shown in Figure 3 and in Appendix C. The two physical CPUs, all memory banks and the HDD are the hardware components measured separately during power measurements. This measurement setup allows to measure the impact of virtualization platforms in a new and refined way.

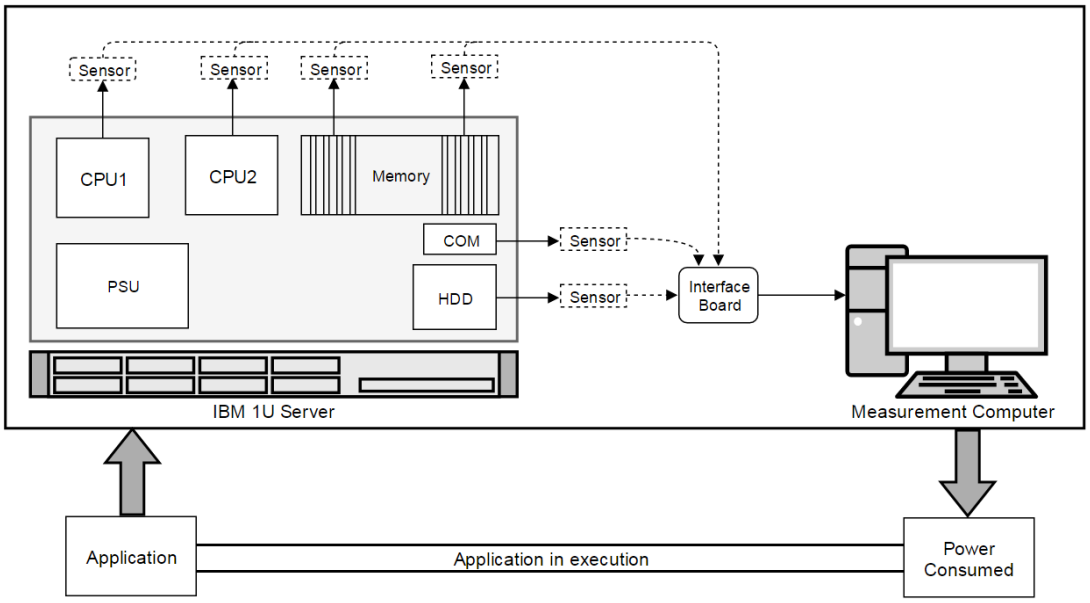
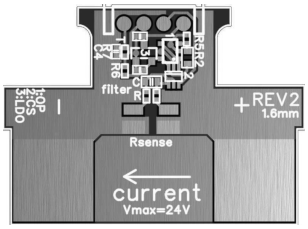
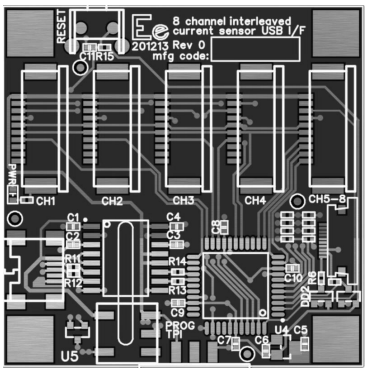


Figure 3: Power Measurement Setup at SEFlab [13]

The power sensors built with Eurocircuits PCBs (Printed Circuit Boards) [48], shown in Figure 4a, are installed onto the motherboard by installing connectors into the component power circuitry. The power sensors are mounted onto the connectors. Next, these power sensors measure the current (I) and the voltage (V) of the connected component such that an accurate power usage (P) is obtained by calculating $P = I * V$. The current (I) is measured with a sense resistor through which the component current flows. The resulting small voltage over this shunt is filtered, amplified and digitized on the power sensor itself. This digitized signal is sent to a second module, the data-acquisition interface board, shown in Figure 4b, which sends the data to the computer by use of an USB-interface. In post processing, the measured current is gained as this digitized signal is converted into the actual current, by use of the full scale voltage, the shunt resistance, and amplification factor. The voltage measured by the power sensor is transported in analog to the data-acquisition interface board, where it is divided by a resistor bridge and also digitized. The measured voltage is gained in a comparable way as it is done with the current. To improve the accuracy of the power measurements, each sensor is calibrated such that the measurement error margins are minimised. Appendix D describes the exact specifications of the measurement boards. Ultimately, the resulting deviation is smaller than 2 percent.



(a)



(b)

Figure 4: (a): Power Usage Measurement Sensor [48]
(b): Data Acquisition Interface Board [48]

5.3 Data Acquisition Software

Data acquisition software written for this research project is custom made, specifically to read out the acquired data stream. The power consumption data is measured by the power sensors and is sent directly to the data-acquisition interface board which in turn sends it to the data acquisition software. The current is sampled 1000 times a second together with the less altering voltage. The data acquisition software calculates the power consumption (P) by multiplying both voltage (V) and current (I) as in $P = U * I$. This power consumption (P) is later appended to a csv-file. The software reduces the sample speed to 10 samples per second by averaging, before writing the data to the .csv-file. The resulting csv-file is later used by the statistical program R to interpret results. Ultimately, this experiment should outline the energy footprint and the power efficiency of each hardware component, CPU, Memory and HDD, running on bare-metal, Xen and Docker. Measurement data acquired during this research is uploaded on the following GitHub repository:

`https://github.com/energy-hypervisor-container-rp2/measurements.git`

5.4 Running the Measurements

Measurements are only executed 30 minutes after the server is booted to prevent other system processes form interfering. Note that there is a power sensor placed on the serial COM port of the server. The small power usage of this COM port will be used to determine the start and stop time of each measurement. By making a reverse serial connection to the ttyusb0 COM port just before executing the synthetic application, a spike of 20 mW is registered. This allows precise registration of an accurate start and stop time, which are of vast importance during the statistical processing phase.

5.5 Server Specifications

Table 2 lists the components present inside the IBM 1U rack-server. This IBM 1U [49] is representable for an in-use data center server. Xen, Docker and the Native Ubuntu OS will run on separate physical hard disk drives so that the libraries for each platform are separated. Furthermore, the server's physical hardware includes hardware virtualization acceleration support (VT-x). The installation of the latest Ubuntu version encountered a few compatibility issues. These issues are explained in Appendix E.

	System Information
Model	IBM System x3550 M4 7914B3G Server (Firmware version 1.9)
Motherboard	IBM 00D3449
Power Supply	IBM Emerson 550W (80+ Platinum Certification)
Processors	2x Intel Xeon E5-2609 v2 @ 2.50 GHz, 8 cores, 6.4 GT/s, 25 MB Cache
Memory	64 GiB (8 GiB x 8 Slots) Hynix DDR3 PC3 1333MHz 14900R, 13-12-B1
HDD	HP 146 GB 2.5-inch SCSI SAS, 3.0 GB/sec, 10,000 RPM

Table 2: IBM 1U Server Specifications [49]

6 Experiments

This chapter explains which synthetic applications were used to benchmark the server. Furthermore, the calculation methodology to acquire the power efficiency is explained.

6.1 Synthetic Applications

The power consumption of hypervisor-based virtualization and Linux containers will be measured using a number of resource intensive applications. Three synthetic applications will be ran, each stressing a different hardware component, being either the processor, memory or the HDD, which is described in Table 3. The following synthetic applications will be used as workload generators on the IBM 1U server:

Hardware Components	**Synthetic Application**
Processors	Intel (MKL) modified LINPACK 11.3.1.002 [23]
Memory	sysbench 0.4.12-1.1 [24]
HDD	Bonnie++ 1.97.1 [25]

Table 3: Hardware components and their synthetic benchmark applications

6.1.1 LINPACK

LINPACK is a synthetic software library that performs a number of linear algebra problems [18]. LINPACK is written in FORTRAN by Jack Dongarra et al. However, Intel released its own version of LINPACK [23] that is optimised for Intel processors. Therefore, the Intel version of LINPACK will be used during this research project.

LINPACK measures how fast the server can solve a dense $N * N$ system of linear equations, $Ax = b$. The algorithm applies Gaussian Elimination with Partial Pivoting, and has a time complexity as stated below [19]:

$$\frac{2}{3}N^3 + 2N^2 + \mathcal{O}(N)$$

The exact performance results are expressed in MFLOPS (Mega Floating Point Operations Per Second) [20]. The LINPACK benchmark script in Appendix F is used during this project. A total of 15 runs will be completed using a problem size where $N = 22350$.

6.1.2 sysbench

sysbench [24] is an open-source benchmark utility that can load the CPU, memory or disk I/O. This research configures sysbench to transfer 333 GiB (2^{30} bytes) over the memory bus. The benchmark script in Appendix G is used during this project.

6.1.3 Bonnie++

Bonnie++ [25] is also an open-source benchmark utility that will be used to stress the HDD. The disk I/O will be measured transferring a test file of 26844 Megabyte, which is the equivalent of 25 GiB. Bonnie++ benchmarks the sequential writes and sequential reads in block speed [51]. The benchmark script in Appendix H is used to benchmark the HDD.

6.2 Calculation Methodology

Since not all hardware components are accountable for power consumption, the following three hardware components will be measured internally: two physical processors, all memory banks and the HDD. Synthetic applications described in the previous chapter will run with a number of compute intensive tasks for each hardware component. Every benchmark is executed between 5 and 15 times to achieve more reliable results within the data set.

Equation 1 expresses how the mean (μ) average power (in Watt) is calculated. The Average Power (AP) is calculated by adding all individual measurements during different benchmark runs n_i, then dividing them by the total number of measurements by the benchmark runs (N). The duration of each benchmark run is approximately 8 minutes for CPU, memory and HDD. The data on power usage of the measured hardware component is collected 1000 times per second during one benchmark run. Therefore, the Equation below for the Average Power $N \approx 5\ (Benchmark\ runs) \times 8\ (Minutes) \times 60\ (Seconds) \times 1000\ (Measurements)$.

$$Average\ Power\ (AP) = \frac{1}{N} \sum n_i \tag{1}$$

Next, Equation 2 expresses how the sample standard deviation (STD) is calculated. The sample standard deviation is used to measure the variability of each individual measurement around the Average Power (AP).

$$Sample\ Standard\ Deviation\ (STD) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (n_i - AP)^2} \tag{2}$$

Next, the total energy (in Joule) can be calculated by multiplying the average power (in Watt) times the total time (in seconds) that the benchmark took to complete. This calculation is shown in Equation 3:

$$Total\ Energy = Average\ Power\ (AP) \times Total\ Time \tag{3}$$

Ultimately, the power efficiency [52] will be calculated, according to the Green List 500 methodology [53] by dividing the average compute power, expressed either in Mega Floating-point Operations Per Second (MFLOPS), Operations Per Second (Ops) or Kilobit Per Second (Kb/s), by the Average Power. This calculation is shown in Equation 4:

$$Power\ Efficiency\ (PE) = \frac{Average\ Compute\ Performance\ (ACP)}{Average\ Power\ (AP)} \tag{4}$$

In order to determine the reliability of Power Efficiency (PE), the sample standard deviation (STD) was used. As power efficiency depends on both Average Compute Performance and Average Power (as is shown in equation 4), standard deviation of Power Efficiency is computed in Equation 5. This formula neglects possible correlations between Average Compute Power and Average Performance.

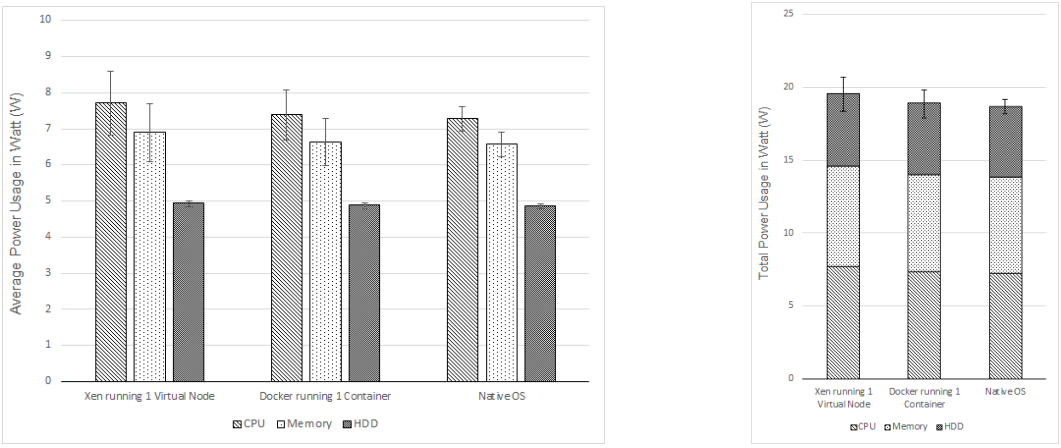
$$\frac{STD_{PE}}{PE} = \sqrt{(\frac{STD_{ACP}}{ACP})^2 + (\frac{STD_{AP}}{AP})^2} \tag{5}$$

7 Results

This chapter presents and attempts to justify the measurement results found during this research. First, the IDLE power consumption of each measured hardware component is discussed to establish a baseline. Next, the CPU, Memory and HDD results are presented and explained.

7.1 IDLE Results

Figure 5 shows the average power usage (in Watt) when the platforms are running IDLE. Xen is running one virtual node while Docker is running one container, both with the same OS kernel as the host system. Native OS in this research represents a patched 64-bit version of Ubuntu Server 15.10. As described in the Setup Section, all platforms run on a separate HDD, such that a fair comparison can be made. The observation time is roughly 15 minutes, resulting in approximately 55,000 sample entries of power usage for each measured hardware component. These measurements have been repeated at least five times. The first half an hour after the server is booted was not measured, in order to minimise system processes from intervening.



(a) (b)

Figure 5: (a): Average Power Usage in Watt (W) when IDLE
(b): Total Average Power Usage in Watt (W) when IDLE
Exact numbers are given in Table 4 and Table 5
Error-bars in all bar-plots indicate the sample standard deviation obtained over all runs

	AP_{CPU}	AP_{MEMORY}	AP_{HDD}	AP_{TOTAL}
Xen	7.70	6.90	4.93	19.53
Docker	7.38	6.63	4.87	18.88
Native OS	7.27	6.56	4.86	18.68

Table 4: Average Power Usage (AP) in IDLE (in Watt) for each Measured Hardware Component

	STD_{CPU}	STD_{MEMORY}	STD_{HDD}	STD_{TOTAL}
Xen	0.88	0.80	0.08	1.19
Docker	0.69	0.66	0.08	0.95
Native OS	0.34	0.34	0.07	0.49

Table 5: STD of the Average Power Usage (AP) in IDLE (in Watt)

On average, the server's total power usage in IDLE was deduced from a power measuring device, where the server roughly consumes 70 Watt of power. This means, hardware components such as the PSU, fans and other components are unmeasured power overhead. Next, Table 4 compares each platform of each hardware component individually, where the power usage is obtained from the internal power sensors. The CPU variable accounts for the total Wattage of physical CPU1 and CPU2. Memory is the total Wattage of all memory banks, while the HDD variable accounts for the total wattage used for the HDD. The TOTAL variable is considered the sum in Watt of CPU, memory and HDD:

$$AP_{TOTAL} = < AP_{CPU} > + < AP_{Memory} > + < AP_{HDD} >$$

The errors are calculated by taking the sample STD of the AP_{CPU}, AP_{MEMORY} and AP_{HDD}. The STD_{TOTAL} is calculated as the square root of variances of the measured hardware components:

$$STD_{TOTAL} = \sqrt{STD^2_{CPU} + STD^2_{MEMORY} + STD^2_{HDD}}$$

It can now be deduced from Table 4 and Figure 5a that all platforms roughly use the same average power in the range of approximately 1 Watt, which can be considered almost negligible on a small scale. Nonetheless, Xen consumes the most power in Watt on average. These IDLE results give a fundamental baseline for the next measurements.

The Ericsson researcher also studied the IDLE usage of Xen, Docker and a native OS [2]. However, this research deploys multiple virtual nodes and containers. This research shows an average power consumption of 123 Watt for native OS, 124 Watt for docker and 128 Watt for Xen. However, the researcher uses a power measuring device, which results in a less refined sensing setup. Differences in results are likely due to the different types of nodes used, which have different characteristics - hence different absolute values. Nonetheless, the Ericsson researcher also states Xen consumes the most power, even though the difference respect to containers is not high [2].

7.2 CPU Results

The IBM 1U server contains two physical processors, each with four cores. Cores [0-3] are registered to the first physical CPU1, while cores [4-7] are registered to physical CPU2. Two experiment scenarios were researched. The first experiment deploys a virtual entity, a Xen virtual node and a Docker container, each allocated four vCPUs pinned to the first four physical cores of the first CPU1. Hence, all cores [0-3] from CPU1 will be allocated to the virtual entities.

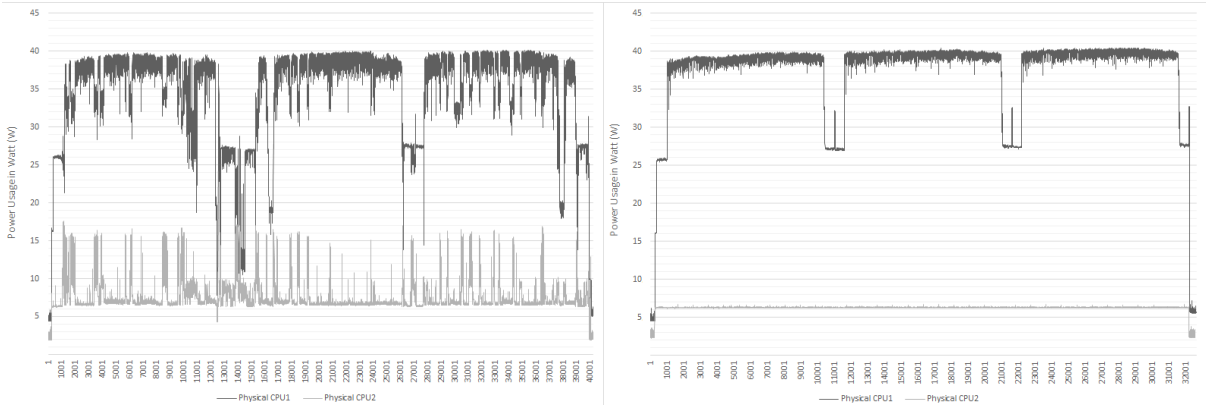
The second research scenario deploys a virtual entity with also four allocated vCPUs. However, this time, each vCPU is pinned to two cores of each physical CPU1 and CPU2 core [2-5]. Hence, cores [2-3] of physical CPU1 and cores [4-5] of physical CPU2 are allocated. This scenario should divide the CPU workload over both physical processors. Appendix I shows the configuration used during this research.

Table 6 shows the power consumption and total duration in seconds during the LINPACK benchmark described in the previous chapter. The average CPU power usage in Watt accounts for the average power consumption of both physical CPU1 and CPU2. Differences in power consumption of both virtualization platforms is not significant on this scale. However, the total duration in seconds varies significantly for the Xen virtual node. On average, there is approximately a 10 second difference in completing the same LINPACK benchmark task for Xen nodes [0-3] and Xen nodes [2-5]. Furthermore, the difference between a Xen virtual node and a Docker container is even more significant. Xen takes approximately 80 seconds longer to complete the same LINPACK benchmark runs compared to a Docker container.

	AP_{CPU}	STD_{CPU}	*Average Duration*	$STD_{AVERAGE\ DURATION}$
Xen cores [0-3]	44.52	3.23	434	3.29
Xen cores [2-5]	46.56	4.24	424	2.77
Docker cores [0-3]	45.78	0.48	348	0.58
Docker cores [2-5]	47.98	0.49	350	0.55

Table 6: Average CPU Power Usage (AP_{CPU}) (in Watt)
Average Duration (in Seconds)

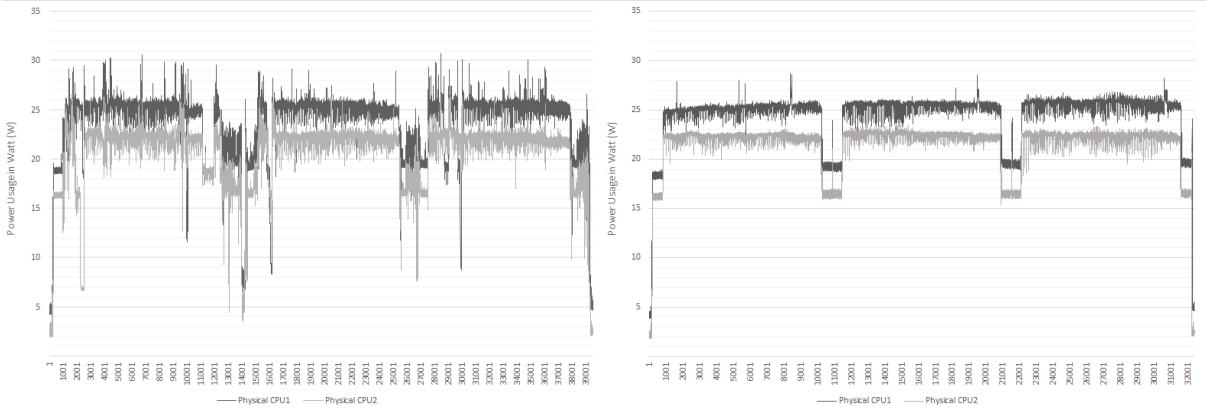
Figure 6 shows the result of the first research scenario. This Figure shows three LINPACK benchmark runs on both an Xen and Docker virtual entity. Both Figure 6a and 6b show the average power usage when cores [0-3] of physical CPU1 are allocated to the virtual entities. It can be observed from Figure 6a that the Xen virtual node is intensively content switching between physical CPU1 and CPU2, compared to a much cleaner CPU utilization of the Docker container, which is shown in Figure 6b. However, the observed behaviour of Xen might as well be the results of the Xen scheduler interfering with the vCPU-pinning configuration. It can also be observed that LINPACK is staging before running the actual benchmark run. The Xen PVHVM node seems to have more difficulty staging down after each benchmark run.



(a) (b)

Figure 6: (a): LINPACK on an Xen node loading cores [0-3]
(b): LINPACK on a Docker container loading cores [0-3]
Shown are the curves of CPU1, cores [0-3]

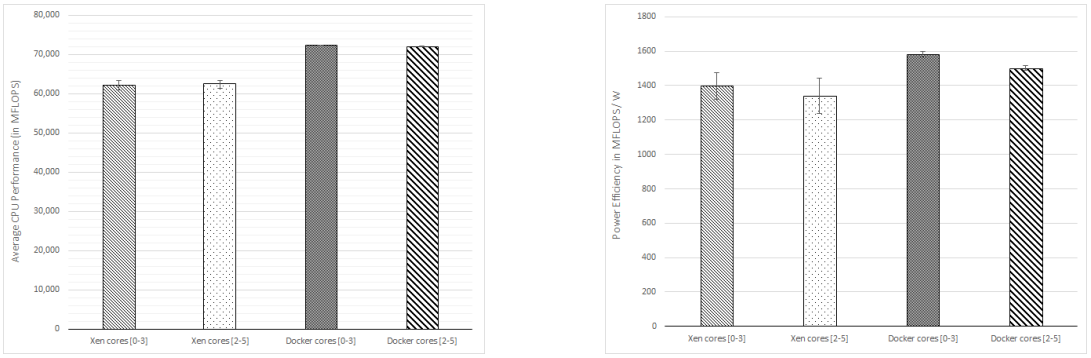
Figure 7 shows the result of the second research scenario. Again, three LINPACK benchmark runs performed on a Xen and Docker virtual entity. However, this time both physical CPU1 and CPU2 are used to divide the workload. The Xen PVHVM virtual node shown in Figure 7a fluctuates more in terms of power usage compared to the Docker container in Figure 7b.



(a) (b)

Figure 7: (a): LINPACK on an Xen node loading cores [2-5]
(b): LINPACK on a Docker container loading cores [2-5]
Shown are the curves of CPU1, cores [2-3] and CPU2, cores [4-5]

Table 7 and Figure 8 show the total CPU performance results during 15 LINPACK benchmark runs. The number of Mega Floating-point Operations Per Second (MFLOPS) done between the two scenarios on the same platform is not that significant. However, The Xen PVHVM virtual node performs approximately 10,000 MFLOPS less than the Docker container. Next, power efficiency of Xen and Docker is expressed according to the two research scenarios. It can be deduced from these results that Docker with CPU core [0-3] configuration is the most efficient. Xen, configured with cores [2-5], is the most inefficient in terms of power efficiency. It should be noted that without resource allocation and isolation, and without a virtualization platform, the server's Native OS consumes on average 73.5 Watt on CPU power, and performs approximately 139,638 MFLOPS.



(a) (b)

Figure 8: (a): Average CPU Performance (ACP_{CPU}) (in MFLOPS)
(b): CPU Power Efficiency (PE_{CPU}) (in MFLOPS / W)
Exact numbers are given in Table 7

	ACP_{CPU}	STD_{ACP}	AP_{CPU}	STD_{AP}	PE_{CPU}	STD_{PE}
Xen cores [0-3]	62,204	1,145	44.52	3.23	1,397	105
Xen cores [2-5]	62,441	929	46.56	4.24	1,341	124
Docker cores [0-3]	72,401	17	45.78	0.48	1,581	17
Docker cores [2-5]	72,075	101	47.98	0.49	1,502	16

Table 7: Average CPU Performance (ACP_{CPU}) (in MFLOPS)
Average CPU Power Usage (AP_{CPU}) (in Watt) Obtained from the previous Table 6
CPU Power Efficiency (PE_{CPU}) (in MFLOPS / W)

As described in the related work section, the IBM research division studied the CPU performance's impact on Docker and KVM [7]. The Intel MKL LINPACK version is used to benchmark the virtualization platforms using a problem size where $N = 45000$, executing 10 benchmark runs. Furthermore, this research allocates all 16 physical cores to the virtual entities, instead of 8 cores. The IBM research division observes similar results. Performance is almost identical on both the Native OS and Docker. This is justified because of the little OS involvement during the execution [7]. However, the KVM performance is markedly worse, as can be seen in Figure 1. This shows the costs of abstracting, thus hiding system information from the execution [7]. Although Xen is not researched by the IBM division, it uses a similar virtualization technology, namely a hypervisor.

One year later, the Ericsson research division also studied the CPU performance impact on Docker and KVM [1] by using a different version of LINPACK [50] with a problem size where $N = 1000$. This study states that the relative differences in performance between the different virtualization platforms are not substantial [1]. The study also claims differences are neglectable with larger values of N [1]. These results do not match with the results from the IBM research division nor from the results acquired during this research. The Ericsson research mentions the IBM research in its related work, but does not compare its results, nor does it elaborate on the different results.

The Ericsson researcher also studied the CPU power consumption of Xen and Docker [2]. This study outlines differences in power usage between the virtualization platforms. These differences might be partly due to a different Xen mode since this was not specified in the Ericsson research. Furthermore, the results are hard to compare as the researcher studied the active power and not the power efficiency.

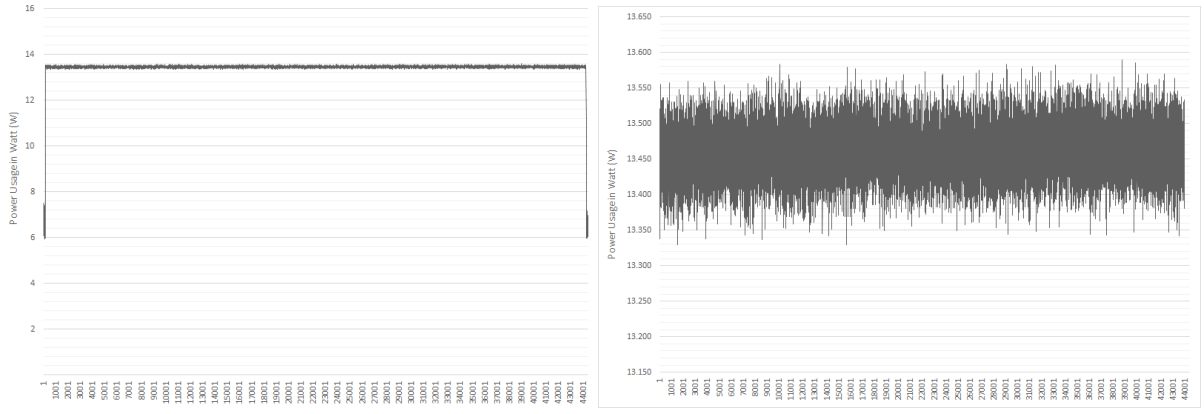
7.3 Memory Results

Table 8 shows the results of the sysbench benchmark described in the previous chapter. Both Xen and Docker are allocated 4 GiB of virtual memory. During the sysbench memory transfer of 333 GiB, the average memory power usage in Watt remained the same on both virtualization platforms. Furthermore, the total time it took to complete the benchmark run differs 1 second, which is considered to be negligible. Native OS is not compared because results have to be derived based on heuristics of the total system memory, which is 16 GiB. This would result in an unfair comparison.

	AP_{MEMORY}	STD_{MEMORY}	*Average Duration*	$STD_{Average\ Duration}$
Xen	13.48	0.03	271	2.50
Docker	13.47	0.03	270	3.48

Table 8: Average Memory Power Usage (AP_{MEMORY}) (in Watt)
Average Duration (in Seconds)

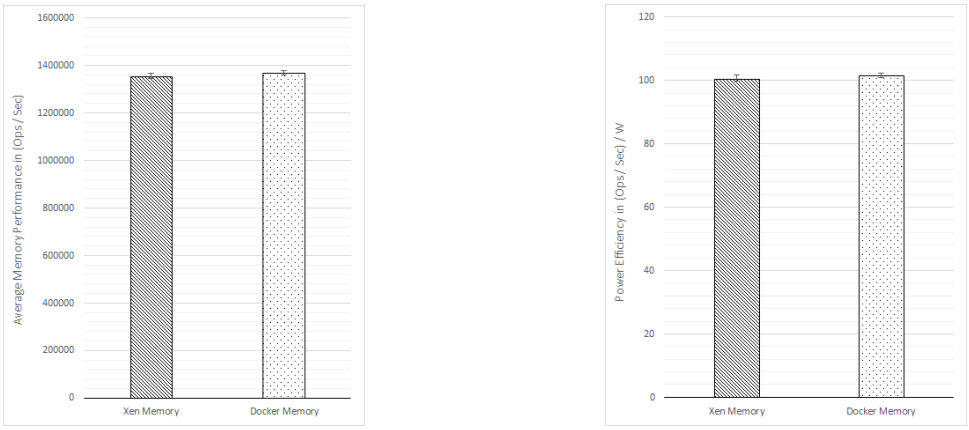
Figure 9a shows the average power usage in Watt. Unlike the CPU benchmark runs, no staging takes place during the memory transfer. Both Xen and Docker have similar trend lines. Figure 9b shows a scoped frame of Figure 9a.



(a) (b)

Figure 9: (a): Average Memory Power Usage (AP_{MEMORY}) (in Watt)
(b): A scoped view on the Memory Power Usage of in Figure 9a
Exact numbers are given in Table 8

Table 9 and Figure 10 show the performance in Operations per second (Ops). On this scale, both Xen and Docker perform almost equal results. Therefore, the power efficiency difference of both platforms is almost negligible.



(a) (b)

Figure 10: (a): Average Memory Performance (ACP_{MEMORY}) (in Ops / Sec)
(b): Memory Power Efficiency (PE_{MEMORY}) (in (Ops / Sec) / W)
Exact numbers are given in Table 9

	ACP_{MEMORY}	STD_{ACP}	AP_{MEMORY}	STD_{AP}	PE_{MEMORY}	STD_{PE}
Xen	1,355,898	12,373	13.48	0.03	100,585	945
Docker	1,368,017	9,669	13.47	0.03	101,634	753

Table 9: Average Memory Performance (ACP_{MEMORY}) (in Ops / Sec)
Average Memory Power Usage (AP_{MEMORY}) (in Watt) obtained from the previous Table 8
Memory Power Efficiency (PE_{MEMORY}) (in (Ops / Sec) / W)

The IBM research division studied the memory performance of KVM and Docker [7] using STREAM [21]. STREAM executes four operations namely: copy, scale, add, and triad [7]. This research uses a 36 GB working set and allocates half of its dedicated cores to the virtual entities, where once a page table entry is installed in the TLB. The IBM researh division shows that the performance on KVM and Docker is almost identical [7]. However, Xen was not studied during this research.

The Ericsson researchers also studied the memory performance of KVM and Docker [1]. This research also uses STREAM to benchmark the platforms. This research observes KVM and Docker perform similar to the native OS. However, the methodology performed in this research was not clear on how much memory was dedicated to each virtual entity. Furthermore, Xen was not taken into account while studying the memory performance of hypervisor-based virtualization platforms.

The power consumption study of the Ericsson researchers also looked at the memory component [2]. This time, Xen was studied and compared with other virtualization platforms. The Ericsson researcher deploys multiple virtual entities and later observes the average power consumption. All virtual entities are stressed by using the sysbench utility, which is also used during this research. However, the researcher was not on its benchmark methodology. Vital information such as memory throughput was not specified. Nonetheless, the author also concludes hypervisor and container-based virtualization behave on average similarly and no clear difference can be noticed in terms of memory power consumption [2].

7.4 HDD Results

Table 10 shows the results of the average power usage in Watt of the HDD. Docker uses approximately the same amount of power as the Native OS. Xen uses approximately 10 milliWatt (mW) more power compared to the Native OS and Docker. However, the difference is almost negligible. It is interesting to notice here the duration in seconds it took to complete the benchmark. 25 GiB of data was written and read on all platforms. Docker completes the same task approximately 10 seconds faster then Xen.

	AP_{HDD}	STD_{HDD}	*Average Duration*	$STD_{Average\ Duration}$
Xen	6.35	0.42	722	1.41
Docker	6.39	0.37	712	0.55
Native OS	6.48	0.24	629	9.14

Table 10: Average HDD Power Usage (AP_{HDD}) (in Watt)
Average Duration (in Seconds)

Figure 11 shows the average power usage of the HDD during a 25GiB Bonnie++ benchmark. The 12V trend line shows the power usage needed to power the mechanical part of the HDD, while the 5V trend line shows the power usage of the circuitry of the HDD.

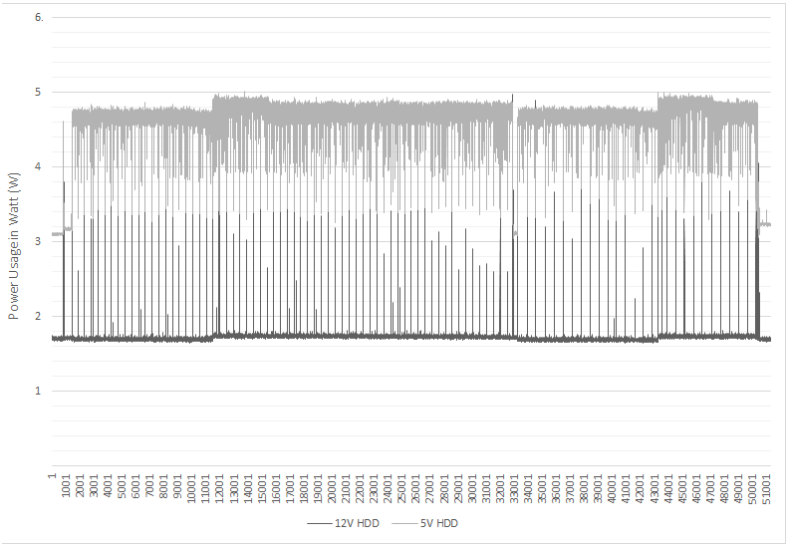
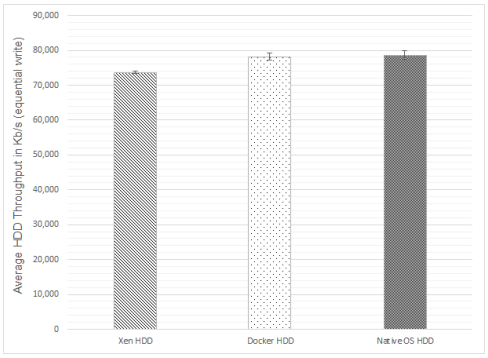
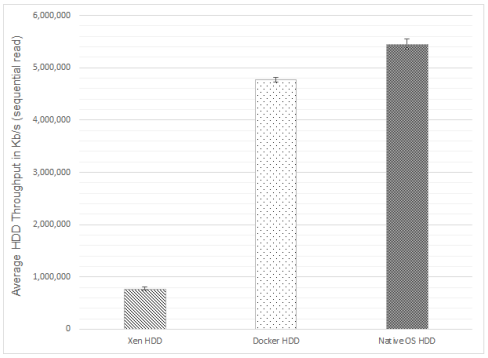


Figure 11: HDD Power Usage (in Watt) while performing a Bonnie++ benchmark

Table 11 and Table 12 show the sequential write and read performance of the HDD when using Bonnie++. Docker performs in both cases the closest to the native OS. Read and write speed can be limited in Docker using cgroups. However, by default, read and write speed is unlimited [56]. By default Xen does not limited the read and write speeds. Xen performs significantly worse in terms of sequential read performance, see Table 12. The Xen PVHVM node was not deployed in logical volume, but on the physical volume as an image file (.img). As described in section 4.2.5, Docker uses UnionFS.



(a)



(b)

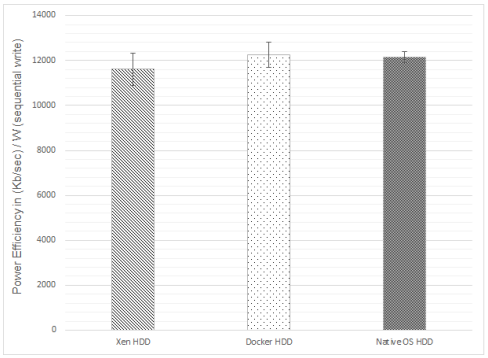
Figure 12: (a): HDD Sequential Write Performance ($APC_{HDD\ Writes}$) (in Kb/s)
(b): HDD Sequential Read Performance ($STD_{HDD\ Reads}$) (in Kb/s)
Exact numbers are given in Table 11 and Table 12

	$ACP_{HDD\ Writes}$	STD_{APC}	$AP_{HDD\ Writes}$	STD_{AP}	$PE_{HDD\ Writes}$	STD_{PE}
Xen	73,720	295	6.35	0.42	11,609	769
Docker	78,292	1,033	6.39	0.37	12,252	728
Native OS	78,670	1,206	6.48	0.24	12,140	487

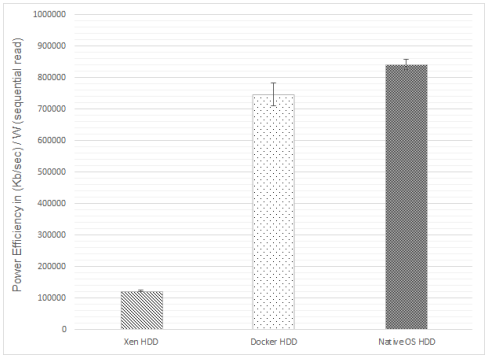
Table 11: Average HDD Write Performance ($ACP_{HDD\ Writes}$) (in Kb/s)
Average HDD Writes Power Usage ($AP_{HDD\ Writes}$) (in Watt) obtained from Table 10
HDD Power Efficiency Sequential Writes ($PE_{HDD\ Writes}$) (in (Kb/s) / W)

	$ACP_{HDD\ Reads}$	STD_{APC}	$AP_{HDD\ Reads}$	STD_{AP}	$PE_{HDD\ Reads}$	STD_{PE}
Xen	777,387	30,846	6.35	0.42	122,423	9,443
Docker	4,773,599	48,148	6.39	0.37	747,042	43,907
Native OS	5,452,535	97,161	6.48	0.24	841,440	34,584

Table 12: Average HDD Write Performance ($ACP_{HDD\ Reads}$) (in Kb/s)
Average HDD Reads Power Usage ($AP_{HDD\ Reads}$) (in Watt) obtained from Table 10
HDD Power Efficiency Sequential Reads ($PE_{HDD\ Reads}$) (in (Kb/s) / W)



(a)



(b)

Figure 13: (a): HDD Power Efficiency Sequential Writes ($PE_{HDD\ Writes}$) (in (Kb/s) / W)
(b): HDD Power Efficiency Sequential Reads ($PE_{HDD\ Reads}$) (in (Kb/s) / W)

Table 11 and Table 12 also shows the power efficiency of both the sequential writes and reads. The power efficiency of the sequential writes is less significant, where Xen performs the worst and Docker and the Native OS approximately the same. Furthermore, Xen performs significantly worse then Docker and the Native OS when comparing the sequential reads due to its bad read performance. Docker has the best power efficient sequential writes. However, if we consider the STD values, the native OS performs within the same values. The native OS has the best power efficient sequential reads, while Xen has the worst performance.

The IBM research create a SAN block storage using ext4 on a 20 TB IBM FlashSystem [58] containing several SSD drives. Therefore, the results of this research will not be compared because of the different storage technologies.

The Ericsson researcher studied the HDD performance of Docker and KVM [1]. Bonnie++ was also used to conduct the experiments with a test file of 25 GiB. This research points out that KVM write throughput is roughly a third and read throughput almost a fifth of the Native OS [1]. The Ericsson study found a mismatch between the results of Bonnie++ and other tools such as sysbench [1]. This suggests that Disk I/O performance estimation can be tricky [1]. Again, Xen was not taken into account. The other Ericsson study [1] does not take the HDD component into account. For reliability, the HDD results should require more research.

8 Conclusions

This research is concluded by answering the following research questions:

How does one measure the power consumption of each hardware component internally in the most accurate way?

This research demonstrates a new refined method of measuring the power consumption of the CPU, Memory and HDD hardware components. Rather than employing a power measuring device, the power consumption of these hardware components is measured internally by placing power sensors between the motherboard and circuits of each measured hardware component. Next, the power consumption is acquired by multiplying both voltage and current. The data on power consumption of these measured hardware components is collected 1000 times per second during one benchmark run. A reverse serial connection, a spike of 20 mW, is registered. This allows precise registration of an accurate start and stop time.

Is there a measurable difference in power efficiency when running an application under a traditional hypervisor-based virtualization versus Linux containers?

First, the results of the IDLE measurements show Xen, Docker and the Native Ubuntu OS roughly use the same average power in the range of approximately 1 Watt, which can be considered almost negligible on a small scale. Nonetheless, Xen consumes the most power in Watt on average. These results show the CPU component is the dominant hardware component.

Next, the CPU power efficiency of Xen and Docker was researched. The server used during this research contains two physical processors, each with four cores. Cores [0-3] are registered to the first physical CPU1, while cores [4-7] are registered to physical CPU2. Two experiment scenarios were researched. The first experiment deploys a virtual entity, a Xen virtual node and a Docker container, each allocated four vCPUs pinned to the first four physical cores of the first CPU1, hence cores [0-3]. Results show that Docker with the CPU cores [0-3] configuration is the most efficient, while Xen, configured with cores [2-5], is the most inefficient in terms of power efficiency. This is mainly due to the performance difference where Xen performs approximately 10,000 MFLOPS less than the Docker container.

Unlike the CPU results, the average memory power efficiency in Watt remained the same on both virtualization platforms. Furthermore, the total time it took to complete the benchmark run differs 1 second, which is considered to be negligible.

Lastly, the HDD power efficiency is divided in sequential write and sequential read results. Xen performs significantly worse than Docker and the Native OS when comparing the sequential reads due to its bad read performance. Docker and the Native OS produce roughly the same values. The power efficiency of the sequential writes is less significant, where Xen performs the worst, and Docker and the Native OS approximately the same. For reliability, the HDD results should require more research.

9 Future Research

New research can be conducted to investigate the impact of power efficiency on other Xen virtualization modes, such as PV, HVM and PVH mode. Furthermore, the impact of power efficiency on multiple virtual nodes and containers can be studied. One can deploy a finite number of virtual entities to its maximum deployment density, such that a power efficiency curve can be mapped. Moreover, real-life applications can be investigated and evaluated.

The Ericsson research team studied the power consumption of the network component [1]. It would be interesting to research the power efficiency of this component using the more refined methodology used in this research. Lastly, other virtualization platforms, such as LXC, KVM and VMware, could be investigated.

Acknowledgements

I would like to thank Arie Taal and Paola Grosso from the System and Network Engineering (SNE) research group for providing the research, the help during the research and reviewing my work. I also would like to thank Erik Hoektra from the HvA, who is also responsible for the SEFlab, for facilitating me at the laboratory, making useful suggestions and the clear guidance.

References

[1] R. Morabito, J. Kjallman, and M. Komu, "Hypervisors vs. Lightweight Virtualization: a Performance Comparison", Ericsson Research, 2015 IEEE International Conference on Cloud Engineering http://metrics.it.uc3m.es/wp-content/uploads/ic2e.pdf

[2] R. Morabito, "Power Consumption of Virtualization Technologies: an Empirical Investigation", December 2015 http://arxiv.org/ftp/arxiv/papers/1511/1511.01232.pdf

[3] Cisco Global Cloud Index, "Forecast and Methodology", 2014 – 2019, 2015 http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.pdf

[4] KVM (Kernel Virtual Machine) http://www.linux-kvm.org/page/Main_Page

[5] J. Philipps, "Ubuntu Xen", April 2015 https://help.ubuntu.com/community/Xen

[6] VMware vSphere https://www.vmware.com/products/vsphere

[7] IBM Research Division, "An Updated Performance Comparison of Virtual Machines and Linux Containersm", July 2014 http://domino.research.ibm.com/library/cyberdig.nsf/papers/0929052195DD819C85257D2300681E7B/$File/rc25482.pdf

[8] dotCloud, "What is Docker?", 2015 https://www.docker.com/what-docker

[9] Earth System Research Laboratory, "Trends in Atmospheric Carbon Dioxide", November 2015 http://www.esrl.noaa.gov/gmd/ccgg/trends/weekly.html

[10] ENERGY STAR, "Report to Congress on Server and Data Center Energy Efficiency", August 2007 https://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf

[11] European Commission, Institute for Energy, "Code of Conduct on Data Centres Energy Efficiency", October 2008 http://ec.europa.eu/information_society/activities/sustainable_growth/docs/datacenter_code-conduct.pdf

[12] Greening the Cloud http://www.greeningthecloud.nl/

[13] SEFlab, "Software Energy Footprint Lab", 2013 http://www.seflab.com/seflab/

[14] Q. Chen, "Towards energy-aware VM scheduling in IaaS clouds through empirical studies", August 2011 https://staff.fnwi.uva.nl/c.t.a.m.delaat/smartgreen/Qingwen-Chen_thesis.pdf

[15] C. van der Poll, "Resource usage in Hypervisors", August 2015 https://esc.fnwi.uva.nl/thesis/centraal/files/f1542049703.pdf

[16] Ubuntu Manuals, "stress - tool to impose load on and stress test systems" http://manpages.ubuntu.com/manpages/trusty/man1/stress.1.html

[17] X. Peng, Z. Sai, "Virtual machine power measuring technique with bounded error in cloud environments", June 2013 http://www.sersc.org/journals/IJGDC/vol6_no3/6.pdf

[18] LINPACK, http://www.netlib.org/linpack/

[19] Dongarra, Jack J., "Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment", SIGARCH Comput. Archit News, vol. 11, nr 5, December 1983, Pages 22-27 http://doi.acm.org/10.1145/859551.859555

[20] Jack J. Dongarra,Victor Eijkhout, Journal of Computational and Applied Mathematics, "Numerical linear algebra algorithms and software", November 2000, Pages 489–514 http://www.sciencedirect.com/science/article/pii/S0377042700004003

[21] McCalpin, John D, "STREAM: Sustainable Memory Bandwidth in High Performance Computers", http://www.cs.virginia.edu/stream/

[22] fio, "Flexible IO Tester", http://git.kernel.dk/?p=fio.git;a=summary

[23] Zhang Z., "Intel LINPACK Math Kernel Library Benchmarks", November 2015 https://software.intel.com/en-us/articles/intel-mkl-benchmarks-suite

[24] Monty Taylor, Launchpad sysbench, "Overview" 2016 ắrlhttps://launchpad.net/sysbench

[25] Russell Coker, Bonnie++, 2016 http://www.coker.com.au/bonnie++/

[26] J. Schwartz, "Are Containers the Beginning of the End of Virtual Machines", October 2014 https://virtualizationreview.com/articles/2014/10/29/containers-virtual-machines-and-docker.aspx

[27] Susanta Nanda Tzi-cker Chiueh and Stony Brook. "A Survey on Virtualization Technologies. RPE Report", pages 1-42, 2005

[28] Docker, "Understand the architecture" https://docs.docker.com/engine/introduction/understanding-docker

[29] S. J. Vaughan-Nichols, "What is Docker and why is it so darn popular?" Figure 1, August 2014 http://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/

[30] Xen Wiki, "Xen Project Software Overview", April 2015 http://wiki.xen.org/wiki/Xen_Project_Software_Overview

[31] R. Pau, FOSDEM, Benefits of the new Xen paravirtualization mode, February 2013 https://archive.fosdem.org/2013/schedule/event/xen_paravirtualization_mode/attachments/slides/241/export/events/attachments/xen_paravirtualization_mode/slides/241/new_xen_paravirt_mode.pdf

[32] R. Pau, Xenbits, "Performance tuning Xen", November 2013 http://xenbits.xen.org/people/royger/talks/performance.pdf

[33] Xen Project Mailing List, "Low CPU performance on Xen PV VM", 12 January 2016 http://lists.xen.org/archives/html/xen-users/2016-01/msg00053.html

[34] X. Z. Wang et al. "An Interactive Web-Based Analysis Framework for Remote Sensing Cloud Computing", ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume II-4/W2, 2015 http://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/II-4-W2/43/2015/isprsannals-II-4-W2-43-2015.pdf

[35] Linux Containers, November 2015, https://linuxcontainers.org

[36] OpenVZ, 24 December 2015 https://openvz.org/Main_Page

[37] VServer, 22 April 2013 http://linux-vserver.org/Overview

[38] Githb, "lmctfy - Let Me Contain That For You", May 28 2015 https://github.com/google/lmctfy

[39] Github, "Opencontainers, libcontainer", 11 January 2016 https://github.com/opencontainers/runc/tree/master/libcontainer

[40] Cambridge University, "Linux Containers", L. Carata, October 2014 https://www.cl.cam.ac.uk/~lc525/files/Linux_Containers.pdf

[41] Razvan, Docker vs Virtualization, September 2014 http://sleekd.com/servers/docker-vs-virtualization/

[42] Docker, "Hub Image Repository", 2015 https://docs.docker.com/docker-hub/

[43] Docker, "Build your own images", https://docs.docker.com/engine/userguide/dockerimages/

[44] Unionfs, "A Stackable Unification File System", 18 May 2015 http://unionfs.filesystems.org

[45] GitHub, "dockviz: Visualizing Docker Data", November 2015 https://github.com/justone/dockviz

[46] University of Amsterdam, System and Network Engineering Research, "Green IT" https://ivi.fnwi.uva.nl/sne/green-it/

[47] VU University Amsterdam, "Homepage of the Software and Services Research Group", January 2016 http://www.s2group.cs.vu.nl/green-lab/

[48] Eurocircuits, http://www.eurocircuits.com

[49] IBM, Hardware Announcement 113-153, "IBM System x3550 M4 servers", September 2013 http://www-01.ibm.com/common/ssi/rep_ca/3/897/ENUS113-153/ENUS113-153.PDF

[50] J. Burkardt, Florida State University, "LINPACK BENCH", March 2008 http://people.sc.fsu.edu/~jburkardt/c_src/linpack_bench/linpack_bench.html

[51] Ubuntu Manual page, "Bonnie++ - program to test hard drive performance" http://manpages.ubuntu.com/manpages/gutsy/man8/bonnie.8.html

[52] W. Manning, "CompTIA Strata - Green IT", Section 3.3 - Measurements, October 2012

[53] The Green List 500, November 2015, http://www.green500.org/greenlists

[54] Ubuntu Manual page, "sysbench - A modular, cross-platform and multi-threaded benchmark tool" urlhttp://manpages.ubuntu.com/manpages/utopic/man1/sysbench.1.html

[55] S. van Vugt, "Memory and CPU allocation in Xen environments: Optimizing performance", December 2007 http://searchservervirtualization.techtarget.com/tip/Memory-and-CPU-allocation-in-Xen-environments-Optimizing-performance

[56] Heinrich, "What is Docker and How Do You Monitor It?", July 2015 http://axibase.com/docker-monitoring/

[57] Xen Wiki, "Virtualization Spectrum", October 2014 http://wiki.xen.org/wiki/Virtualization_Spectrum

[58] IBM, Implementing IBM FlashSystem 840, July 2015 http://www.redbooks.ibm.com/abstracts/sg248189.html

Appendices

Appendix A Xen PVHVM virtual node settings

```
builder          = "hvm"
boot             = "c"
xen_platform_pci = "1"
pae              = "1"

#all four cores on one physical CPU
cpus             = "0-3"
#two cores on each physical CPU
cpus             = "2-5"

vcpus            =  "4"
memory           =  "4096"

name             = "PVHVM_1"
dhcp             = "dhcp"
vif              = [ ‘mac=00:16:3E:85:F1:B1’ ]
disk             = [ ‘file:/etc/xen/PVHVM_1.img,hdc,w’ ]

stdvga           =  "1"
keymap           =  "en"

on_poweroff      = "destroy"
on_reboot        = "restart"
on_crash         = "restart"
```

Appendix B Docker Deployment Settings

```
#Docker container running on cores [0-3]
docker run -it --cpuset-cpus="0-3" --memory 4G --memory-reservation 4G
ubuntu /bin/bash

#Docker container running on cores [2-5]
docker run -it --cpuset-cpus="2-5" --memory 4G --memory-reservation 4G
ubuntu /bin/bash
```

Appendix C Physical installation of the measurement equipment on the server

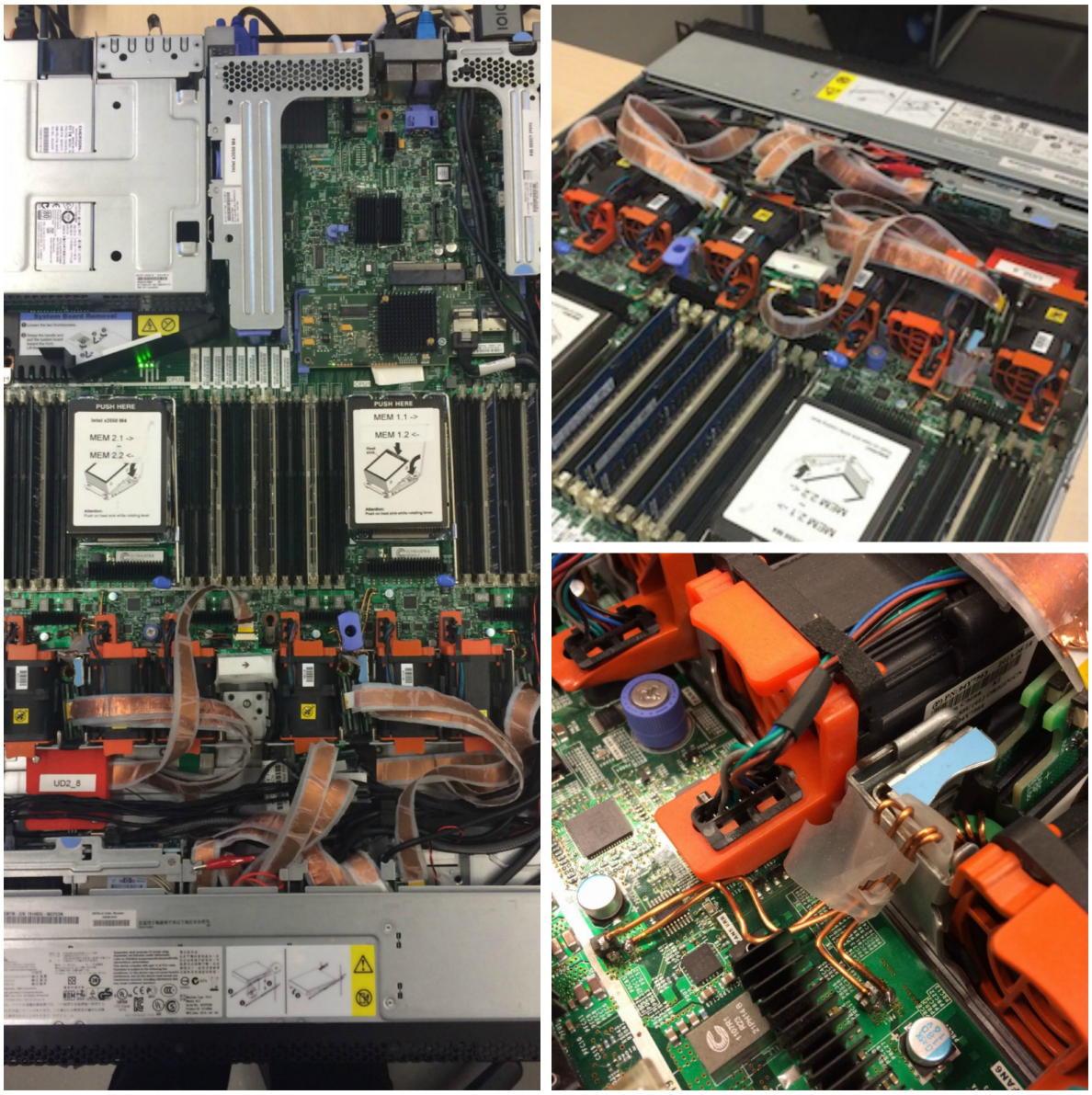


Figure 14: Physical Server

Appendix D Power Sensor Specifications

Flow Measurements	**Values**
Shunt resistor	5 mΩ
Maximum current	20 Ampère
Maximum theoretical deviation	+/- 10%
Deviation after calibration	+/- 2%
3dB Lowpass filter	100 Hz
Differential Amp	50x

Table 13: Power Sensor Flow Measurement

Voltage Measurements	**Values**
Accuracy	+/- 2%
Maximum voltage	20 V

Table 14: Power Sensor Measurement

Sensor Temperature	**Values**
Variable Temperature	20° - 60°
Preciseness	+/- 3°

Table 15: Power Sensor Temperate

Appendix E Install Xen 4.5.1 Hypervisor on IBM x3550 Server

There were some challenges installing the latest 64-bit Ubuntu 15.10 operating system on the IBM x3550 server in BIOS only mode. I was struggling to boot Ubuntu 15.10 x64 Server from a bootable USB memory device on an IBM X3550 M4 7914 server.

I figured that this might be a bug, so I upgraded to the latest server firmware version 1.91 and Integrated Management Modules (IMM) 5.20. However, this was without success. Adding the Generic boot option and placing it on top of the boot order did not work either. Also the removal of the `/USBSTICK/EFI/boot/bootx64.efi` file did not work.

I was forced to apply the following work-around: PXE boot Ubuntu 15.10 from another machine (tftpd32). Next, I installed OpenSSH along with the kernel in order to login remotely because the screen remained black.

The behaviour seems like a bug, but could just be the result of a bad combination of platform and an uncertified operating system. According to ServerProven[1], Ubuntu 15.x is not certified for any of the IBM 3650 M4 machine types (14.x and prior appear to be). Red Hat Enterprise Linux 7 did however boot from a USB-stick.

[1] `https://www.ibm.com/developerworks/community/forums/html/topic?id=1b2be66e-5a33-404e-9c16-b64a03adad3a`

Appendix F LINPACK Benchmark Script

```
#!/bin/bash
#Author: J. van Kessel based on J. van der Poll benchmark scripts
#11-01-2016, Amsterdam
#CPU Benchmark script

#Variables
RUNTIME=480                             #Runtime in seconds
BENCHMARK=1                             #Benchmark time in seconds
BEGIN_TIME=$(date +%s)                  #begin Time
CURRENT_TIME=$(date +%s)                #Current time in seconds
END_TIME=$((CURRENT_TIME + $RUNTIME))   #Total runtime in seconds
ROUNDS=0                                #number of rounds variable

gnome-terminal -e 'screen /dev/ttyUSB0 115200'

printf "Loading the following variables: \n\n"
printf "PID:           "$$" \n"
printf "Current Date:  $(date) \n"
printf "Begin Time:    "$BEGIN_TIME" (in seconds) \n"
printf "End Time:      "$END_TIME" (in seconds) runs "$RUNTIME" seconds \n\n"

#Running the synthetic LINPACK 11.3.1.002 CPU application:
while [ "$CURRENT_TIME" -lt "$END_TIME" ]
do
    printf "Loading the system... \n"
    l_mklb_p_11.3.1.002/linpack/./xlinpack_xeon64 -i linpack_settings.txt
    CURRENT_TIME=$(date +%s) #update current time variable
    let ROUNDS="$ROUNDS"+1 #update number of rounds done
    if [ "$?" -eq "0" ] #error handeling
      then
        printf "\n Number of benchmarks rounds done: "$ROUNDS" \n\n"
      else #error handeling
        printf "Benchmark Failed! \n\n"
          exit 1
    fi
done
printf "Total spent `expr "$CURRENT_TIME" - "$BEGIN_TIME"` seconds. \n"
printf "ALL BENCHMARKS ARE COMPLETED. \n"
```

```
Sample Intel(R) Optimized LINPACK Benchmark data file (lininput_xeon64)
Intel(R) Optimized LINPACK Benchmark data
1                       # number of tests
22350                   # problem sizes
22350                   # leading dimensions
3                       # times to run a test
1                       # alignment values (in KBytes)
```

Appendix G sysbench Benchmark Script

```
#!/bin/bash
#Author: J. van Kessel based on J. van der Poll benchmark scripts
#11-01-2016, Amsterdam
#Memory Benchmark script
clear

#Variables
RUNTIME=480 #Runtime in seconds
BENCHMARK=1 #Benchmark time in seconds
BEGIN_TIME=$(date +%s) #begin Time
CURRENT_TIME=$(date +%s)  #Current time in seconds
END_TIME=$((CURRENT_TIME + $RUNTIME)) #Total runtime in seconds
ROUNDS=0 #number of rounds variable

gnome-terminal -e 'screen /dev/ttyUSB0 115200'

printf "Loading the following variables: \n\n"
printf "PID:           "$$" \n"
printf "Current Date:  $(date) \n"
printf "Begin Time:    "$BEGIN_TIME" (in seconds) \n"
printf "End Time:      "$END_TIME" (in seconds) runs "$RUNTIME" seconds \n\n"

#Running the synthetic sysbench application with the following switches:
while [ "$CURRENT_TIME" -lt "$END_TIME" ]
do
printf "Loading the system... \n"
sysbench
    --test=memory
    --memory-block-size=1K
    --memory-scope=global
    --memory-total-size=350G
    --memory-oper=write
run

CURRENT_TIME=$(date +%s) #update current time variable
let ROUNDS="$ROUNDS"+1 #update number of rounds done
if [ "$?" -eq "0" ] #error handeling
  then
     printf "\n Number of benchmarks rounds done: "$ROUNDS" \n\n"
  else #error handeling
 printf "Benchmark Failed! \n\n"
     exit 1
fi
done
printf "Total spent `expr "$CURRENT_TIME" - "$BEGIN_TIME"` seconds. \n"
printf "ALL BENCHMARKS ARE COMPLETED. \n"
```

Appendix H Bonnie++ Benchmark Script

```
#!/bin/bash
#Author: J. van Kessel based on J. van der Poll benchmark scripts
#11-01-2016, Amsterdam
#HDD Benchmark script
clear

#Variables
RUNTIME=480    #Runtime in seconds
BENCHMARK=1 #Benchmark time in seconds
BEGIN_TIME=$(date +%s) #begin Time
CURRENT_TIME=$(date +%s)  #Current time in seconds
END_TIME=$((CURRENT_TIME + $RUNTIME)) #Total runtime in seconds
ROUNDS=0 #number of rounds variable

gnome-terminal -e 'screen /dev/ttyUSB0 115200'

printf "Loading the following variables: \n\n"
printf "PID:           "$$" \n"
printf "Current Date:  $(date) \n"
printf "Begin Time:    "$BEGIN_TIME" (in seconds) \n"
printf "End Time:      "$END_TIME" (in seconds) runs "$RUNTIME" seconds \n\n"

#Running Bonnie++
#24600MiB is 25GiB disk Throughput used in this benchmark in combination with 4GB memory
while [ "$CURRENT_TIME" -lt "$END_TIME" ]
do
printf "Loading the system... \n"
bonnie++ -d /tmp -r 4096 -s 26850 -u root
CURRENT_TIME=$(date +%s) #update current time variable
let ROUNDS="$ROUNDS"+1 #update number of rounds done
if [ "$?" -eq "0" ] #error handeling
  then
     printf "\n Number of benchmarks rounds done: "$ROUNDS" \n\n"
  else #error handeling
 printf "Benchmark Failed! \n\n"
     exit 1
fi
done

printf "Total spent `expr "$CURRENT_TIME" - "$BEGIN_TIME"` seconds. \n"
printf "ALL BENCHMARKS ARE COMPLETED. \n"
```

Appendix I Xen CPU Pinning

Xen CPU topology used during this research [55].

```
#Xen Dom0 allocated to use and switch between all 8 vCPUs/ CPU cores
Name                                ID  VCPU  CPU  State   Time(s) CPU Affinity
Domain-0                             0     0    0    -b-      77.4  all / all
Domain-0                             0     1    1    -b-      74.4  all / all
Domain-0                             0     2    2    -b-      80.7  all / all
Domain-0                             0     3    3    -b-      71.4  all / all
Domain-0                             0     4    4    -b-      81.7  all / all
Domain-0                             0     5    5    -b-      90.7  all / all
Domain-0                             0     6    6    -b-      94.2  all / all
Domain-0                             0     7    7    r--     542.5  all / all

#4vCPUs used for pvhvm node. Core [0-3] are pinned to vCPU [0-3] and allocated
Name                                ID  VCPU  CPU  State   Time(s) CPU Affinity
pvhvm                                1     0    0    r--    1622.8  0-3 / 0-3
pvhvm                                1     1    1    r--    1622.5  0-3 / 0-3
pvhvm                                1     2    2    r--    1622.2  0-3 / 0-3
pvhvm                                1     3    3    r--    1622.6  0-3 / 0-3

#4vCPUs used for pvhvm node. Core [2-5] are pinned to vCPU [2-5] and allocated
Name                                ID  VCPU  CPU  State   Time(s) CPU Affinity
pvhvm                                1     2    2    r--    1843.3  2-5 / 2-5
pvhvm                                1     3    3    r--    1843.4  2-5 / 2-5
pvhvm                                1     4    4    r--    1843.2  2-5 / 2-5
pvhvm                                1     5    5    r--    1843.4  2-5 / 2-5
```