

UMEÅ UNIVERSITY  
Department of Computer Science  
Report

March 25, 2024

5DV143  
**Master Thesis**

Power Profiling: Understanding the Impact of  
CPU Workloads on Container and Virtual  
Machine Power Efficiency

Johan Huusko

**Course coordinator**  
Henrik Björklund  
**Supervisor**  
P-O Östberg  
**External Partner**  
Trafikverket

## 1 Abstract

Data centers power consumption represent a substantial amount of the global power consumption. While hardware has improved over the years, this study focuses on the software side of optimization, looking at the power consumption differences in containers and virtual machines.

Through testing, this project reveals that there is a significant difference in the power consumption of containers and virtual machines. The type of application load imposed on the CPU is a critical factor that significantly changes the power consumption characteristics of the virtualization technologies.

In purely CPU-bound tasks, containers exhibit a marginal edge, in being 0.6% more efficient than virtual machines. When `syscalls` are being invoked, the results show that virtual machines are the more energy efficient choice by a margin. As the workload tends more towards that of I/O operations, the containers are once again the most energy efficient option.

Even though this study shows that one can reduce power consumption by replacing virtualization technologies with each other, more prevalent methods exists that reduce the power consumption even further than what this project has managed.

## Acknowledgements

The end result of this project is, as with many other projects, a collaboration. Before the report starts, I would like to thank some of the individuals that have helped me through this process.

Thank you P-O for the time spent with me in meetings, guiding me through how to properly gather data, display and analyze data in a system that might not be as deterministic as I had thought. Without your insights, I know that my results would have been very different.

I would also like to thank Trafikverket and especially my external advisor, Pontus Blomqvist. Thank you, Pontus, for the discussions we had during our meetings. Your insight in how these systems are used in production was very valuable when writing the report.

Lastly, I would like to thank Tomas Forsman. A good testing environment was key when generating these results. I would not have been able to draw as many conclusions as I did without your help. Thank you for sharing your knowledge of how operating systems work, I did manage to avoid a few caveats during the project thanks to you.

---

**Contents**

<b>1</b>	<b>Abstract</b>	<b>1</b>
<b>2</b>	<b>Introduction</b>	<b>6</b>
<b>3</b>	<b>Problem Description</b>	<b>7</b>
<b>4</b>	<b>Literature study</b>	<b>9</b>
4.1	Data centers . . . . .	9
4.2	Power . . . . .	10
4.3	Virtualization . . . . .	10
4.4	Hypervisor . . . . .	10
4.5	Virtual environments . . . . .	11
4.5.1	Bare-Metal . . . . .	11
4.5.2	Virtual Machines . . . . .	12
4.5.3	Containers . . . . .	12
4.6	CPU Utilization . . . . .	14
4.7	Operating System Noise . . . . .	14
4.8	RAPL - Running Average Power Limit . . . . .	15
4.9	Dynamic Power Consumption . . . . .	15
4.9.1	Dynamic Voltage and Frequency Scaling (DVFS) . . . . .	16
4.10	Security . . . . .	17
<b>5</b>	<b>Testing Methodology</b>	<b>20</b>
5.1	Methods . . . . .	20
5.2	Testing Hardware . . . . .	21
5.2.1	Orbit . . . . .	21
5.2.2	Consumer Grade CPU . . . . .	21
5.3	Tools . . . . .	22

---

5.3.1	Data Gathering . . . . .	22
5.3.2	Workload Generator . . . . .	22
5.4	Pinning Processes . . . . .	23
5.5	Throughput . . . . .	24
5.6	Technologies . . . . .	24
<b>6</b>	<b>Results</b>	<b>26</b>
6.1	Orbit - General results . . . . .	26
6.1.1	Power Consumption Baseline . . . . .	29
6.1.2	CPU Tests . . . . .	30
6.1.3	CPU Tests with System Calls . . . . .	34
6.1.4	CPU Tests with I/O . . . . .	36
6.1.5	Throughput . . . . .	38
6.2	Tests Performed on Desktop PC . . . . .	39
6.2.1	CPU tests . . . . .	39
6.2.2	CPU tests with <code>syscalls</code> . . . . .	40
6.2.3	CPU tests with I/O . . . . .	41
6.2.4	General Thoughts on Desktop PC . . . . .	42
6.3	Data Transfer Power Consumption . . . . .	43
6.3.1	Hardware . . . . .	43
6.3.2	Test Results . . . . .	44
6.4	System on a Chip . . . . .	44
<b>7</b>	<b>Final Thoughts</b>	<b>45</b>
7.1	RQ1 . . . . .	45
7.2	RQ2 . . . . .	45
7.3	RQ3 . . . . .	46
7.4	General Final Thoughts . . . . .	46

<b>8</b>	<b>Future work</b>	<b>47</b>
<b>A</b>	<b>Results</b>	<b>49</b>
A.1	Orbit . . . . .	49
A.2	Bare metal . . . . .	50
A.2.1	CPU load without system calls . . . . .	50
A.2.2	CPU load with system calls . . . . .	53
A.2.3	CPU load with I/O . . . . .	56
A.3	Container . . . . .	59
A.3.1	CPU load without system calls . . . . .	59
A.3.2	CPU load with system calls . . . . .	62
A.3.3	CPU load with I/O . . . . .	65
A.4	Virtual Machine . . . . .	68
A.4.1	CPU load without system calls . . . . .	68
A.4.2	CPU load with system calls . . . . .	71
A.4.3	CPU load with I/O . . . . .	74
A.5	Consumer grade CPU . . . . .	76

## 2 Introduction

Data centers are responsible for an increasing amount of the world's total energy consumption. Some estimate that data centers consume about 3% of the world's total electricity production, with predictions of it rising to 10% in 2030 [1]. In 2014, data centers in the United States consumed 70 billion kWh of energy, which is enough to power 6 million homes [15]. With the amounts of energy being consumed, it is key that data centers use this energy efficiently.

Power Usage Effectiveness (PUE) is the industry standard of measuring the power effectiveness of a data center. It is a ratio of the total power consumption of a data center over the power consumption of the IT equipment. A PUE of 1.0 means that all the energy consumed by the data center is used by the servers. Some studies show that the average data center PUE is about 1.8, meaning that for every 1 Watt of power used by the IT equipment, 0.8 Watt is used on keeping the data center running. Modern data centers manage to reach levels as low as 1.036, which is a huge improvement in efficiency [13]. Most of the improvements in PUE come from improvements in the efficiency of the cooling hardware and power delivery system used in data centers.

Long term, PUE has been improving at a steady rate [17] with Google reporting a PUE of 1.10 across all their large-scale data centers [8]. This is getting close to the theoretically perfect PUE of 1.0. Hopefully, in due time, we reach a point where almost all the energy consumed by data centers is consumed by the IT equipment alone.

With PUE improving for each year, looking at the software side of data centers might be the next step to decreasing power consumption in data center environments. With the amounts of energy being consumed, only a few percentages of energy savings could lead to huge cost and environmental savings for the organization running the data center.

Most organizations that are running a data center for serving applications to customers use some kind of virtualized abstraction level above the physical hardware. This is, most of the time, done by hosting applications through containers or virtual machines. These virtualization technologies are the first layers of software on top of the hardware. Increasing the overall power efficiency of the lowest level of the software stack should decrease power consumption of the whole system without affecting the end user experience.

This project will look at the differences' in energy consumption between two different virtualization technologies. The two technologies that will be investigated are virtual machines (VMs) and containers. A short literature study of the security implications of the two technologies will also be included, since this was requested by the external partner.

### 3 Problem Description

Virtualization technologies have become a crucial part of data centers and cloud infrastructures. It plays an integral role in optimizing server utilization, allowing for multiple workloads on a single physical host. Whether a data center is hosting the infrastructure for containers or virtual machines, it requires some kind of virtualization layer to be able to feed the virtual hosts with physical hardware.

The primary objective of this research is to compare the power consumption characteristics of virtual machines and containers under various workloads. Understanding the power consumption characteristics of these two environments under different types of application workloads might give organizations the ability to choose a more power efficient virtualization technology without impacting the performance of the application.

The focus of the project extends beyond a simple performance benchmark to also look at the differences these virtualization technologies have on the power efficiency under diverse workloads. This should be able to give more context to anyone looking to make informed decisions on the adoption of virtualization technologies that meet performance requirements but also align with sustainability goals.

Not much research has gone into looking at how different virtualization technologies change the energy usage of data centers. Pure performance benchmarking between the different environments exists. With the rising demand of reducing energy needs in data center environments, this could be one solution to that problem.

This project aims to answer the following research questions:

- RQ1. How does the power consumption of containers compare to that of virtual machines?
- RQ2. Does the type of application change the power consumption characteristics?
- RQ3. What are the implications of considering a more power efficient virtualization solution in terms of application throughput?

This will be done through testing of various virtualization technologies, including Docker containers, a virtual machine environment using KVM and, as a baseline power consumption, the application running without any virtualization technology, referred to as bare metal.

Before writing software that can test and expose the underlying virtualization technologies, it is important to create an understanding of how these technologies work and what types of software or hardware solutions exist that can be used to measure power consumption in a running server.

A literature study will be performed where the virtualization technologies will

be investigated together with tools that can be used to streamline the testing procedure.

## 4 Literature study

### 4.1 Data centers

Data centers serve as backbones of the cloud infrastructure. Depending on the type of data center, it could be used for storage, processing or general management of digital data. Data centers are generally divided into categories; enterprise data centers, public cloud data centers and managed data centers [18]. Even though they differ in how they are used, most of them share the same key components:

- Servers: Hardware units that house the physical components responsible for processing data and executing tasks. Often organized in racks.
- Storage: With the ever-increasing amount of data being generated, data centers might house vast storage system to accommodate the data.
- Networking: A data center needs a robust and scalable network infrastructure.
- Power and cooling systems: Hardware inside the servers need to be kept at operating temperature to keep efficiency high and hardware from breaking. With the amounts of energy being consumed in data centers, there is also a significant amount of heat being generated.

All of these part of the data center require electrical energy to function correctly. With these key components, the PUE is a measurement of the ratio of the power consumption of the server, storage and networking in relation to that of the power and cooling systems. Since this project is comparing the power consumption of different software solutions, the PUE value would not change, and is therefore useless in this project as a metric of efficiency. Instead, this project will look at the pure power consumption differences between the different virtual environments.

Reducing the overall power consumption of data center has been a hot research topic in recent years due to their enormous power consumption. A reduction in power consumption leads to less environmental impact and reduces the overall cost of running a data center. Work has been done looking at the power to performance tradeoffs of different power reducing techniques, for example DVFS [12].

There has also been work done looking at optimizing the scheduling used in a container environment from a holistic view. Replacing the default Kubernetes scheduler showed a 10–20% reduction in total power consumption of the cluster used in testing [17], which is a huge saving.

## 4.2 Power

Two main concepts used through this project are power and energy. Power is the rate at which work is performed, while energy is the total amount of work performed over a specific timer period. The relationship between these can be expressed as:

$$E = PT$$

where  $E$  is the system's energy consumption measured in Joules. The Power ( $P$ ) is measured in *Watt* and the period of time ( $T$ ) is measured in seconds.

Power is the momentary use of electricity, while energy is the total amount of electricity used in a given time frame. The unit for power in this report is Watt, while the unit used for energy is W/s if nothing else is specified.

Understanding the difference between these two concepts is important when communicating about power and energy consumption.

## 4.3 Virtualization

Virtualization is a process that allows for more efficient utilization of physical computer hardware and is the foundation of cloud computing [19].

Virtualization is done by abstracting the underlying physical resources of a single machine and dividing them into multiple virtual machines (VMs). Each virtual machine runs its own operating system and is able to behave like an independent machine.

Containers use virtualization differently from virtual machines. Containers use a form of operating system virtualization. They do this by leveraging features from the host operating system to isolate processes and control the processes' access to physical resources [6].

Understanding the difference in how virtualization is achieved in these two technologies is needed to create an understanding of why the power consumption might differ between containers and virtual machines.

## 4.4 Hypervisor

Hypervisors are a central part of today's virtualization technologies. They are software, firmware or hardware solutions that create, coordinate and run virtual machines. They are the abstraction layer between the virtual machines and the physical hardware.

With the emergency of cloud computing, hypervisors became an invaluable tool to efficiently utilize the physical hardware found in data centers.

Hypervisors make virtualization possible by translating requests between the physical and virtual resources. There are two main types of hypervisors, these are referred to as Type 1 and Type 2. They differ in the following ways:

	<b>Type 1</b>	<b>Type 2</b>
Interaction	Directly with the underlying hardware	With underlying hardware through the host Operating System
Execution	Runs directly on the host hardware.	Typically runs as software on top of a traditional Operating System
Allocation	Allocates resources directly.	Hardware resource allocation has to be negotiated with the host Operating System
Performance	No resource negotiation with dedicated underlying resources.	Can only use resources that the Operating System is willing to provide.
Isolation	No shared layer means that Type 1 is inherently more secure.	The host OS is a shared layer between all virtual machines. If the host is compromised, the whole system might be.
Setup	Can be deployed in a number of ways.	Needs a host operating system to be able to run.

**Table 1:** Some of the differences between a Type 1 and Type 2 hypervisor.

Hypervisors are a central part in enabling high performance in virtual environments. Therefore, Type 2 hypervisors are generally only used in instances where performance is not a key interest.

When investigating the power consumption of a virtual machine, selecting a suitable hypervisor is a requirement. In theory, the more efficient the hypervisor, the less power the virtualization layer should consume. Since this research is looking at data center solutions, a type 1 hypervisor should be used in testing.

## 4.5 Virtual environments

Cloud infrastructures generally use some kind of virtual environment to increase the overall utilization of the servers. Currently, the two most popular are containers and virtual machines. The environment changes the way applications are designed and deployed.

### 4.5.1 Bare-Metal

The traditional way of running an application. There is no virtualization present, which means that the physical server is most often used by a single

user. Since there is no virtualization present, multiple applications on the same server have to share all resources, including the Operating System. Relocating applications can only be done to a location with all the runtime requirements already present. For this project, a bare metal baseline will be used where tests are performed to give an idea of how the virtual environment changes the power consumption.

### 4.5.2 Virtual Machines

Virtual Machines creates a virtual environment where physical servers can be emulated. Since hardware is what is being virtualized, an Operating System is installed on the virtualized hardware. This means that Operating Systems can be co-located to other servers running the same virtualization technology. It also creates a level of isolation on the Operating System level, where an Operating System is not ware of others running on the same physical hardware. Applications deployed using Virtual Machines come bundled with all the runtime requirements present. Virtual Machines are able to scale horizontally but come with the cost of additional overhead.

### 4.5.3 Containers

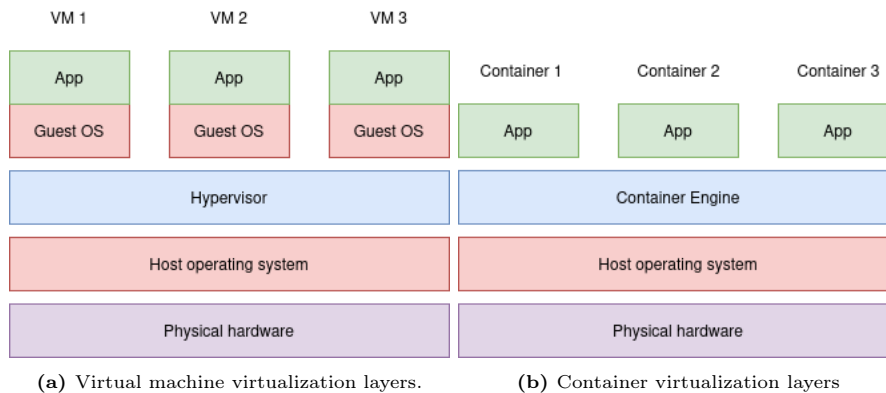
Containers are a standard software unit that comes packages with the code and dependencies required to run an application. Since containers do not contain an Operating System, they are smaller and more lightweight than Virtual Machines. Containers enable horizontal scaling of applications with less overhead than Virtual Machines. Deployment of containers is usually managed with an orchestration software that handles the life cycle and communication between containers.

General differences in how virtual machines and containers are handled can be found in [Table 2](#).

	Virtual Machine	Container
Management	Depending on the amount of Virtual Machines managed, the need for a management might differ. VMs can be managed completely automatically or manually depending on needs or scale.	The design principle of containers makes manual management a nightmare. They need a container management software at scale.
Scaling	Depending on the application, horizontal scaling of virtual machines might or might not be feasible. Vertical scaling can be done by adding more resources.	Containers are by their nature very good at scaling horizontally if the underlying design architecture supports it. Most container technologies support vertical scaling.
Isolation	By nature, VMs are isolated from each other. Management software can be used to further isolate VM's inside a server cluster. The type of hypervisor used changes the level of isolation.	Containers are isolated through the host operating system. On Linux, this is done through cgroups.
Deployment	Applications need to have a guest OS to run inside. This usually needs installation and setup.	Instant deployment of the application if the underlying support is present. Runtime libraries included in the container.

**Table 2:** Differences in how containers and virtual machines are managed and operate.

Figure 1 shows the differences in the virtualization layers between containers and virtual machines. Containers should have less overhead since one of the layers is absent, the Guest OS. Note that the virtual machine solution does not necessarily need a host operating system. This project will, however, use a host operating system between the physical hardware and the hypervisor.



**Figure 1:** Architectural differences between virtual machines and containers.

## 4.6 CPU Utilization

CPU utilization is a measurement of how much time the CPU spends executing processes. The operating system is responsible for doing the calculations, which means that the implementation could be platform specific. Linux schedules a special *idle* task whenever there are no other, higher priority, tasks to be scheduled [5]. It then keeps track of how much time the *idle* task has spent being executed. This can then be used together with the total CPU time elapsed to calculate a CPU utilization.

CPU utilization can be calculated using the following formula:

$$\text{CPU utilization} = \frac{\text{idle task time}}{\text{Total CPU time}}$$

CPU utilization is therefore a measurement of how much time processes are given access to the CPU. It does not mean that the CPU is actively executing tasks. The CPU will be considered utilized while it is waiting for memory access or data from other hardware components.

Since we know that the CPU uses more power the more it is utilized [12], a more lightweight virtualization layer should result in an overall lower CPU utilization and power consumption.

## 4.7 Operating System Noise

Operating system noise encapsulates the time spent by a CPU executing instructions not belonging to a given application task assigned to that CPU while the task is ready to run [14].

Even in scenarios where no other tasks are scheduled on a specific CPU core, hardware induced noise can be found. This is most often caused by hardware stalls, which can be an effect of shared resources. Hyper-threading or other similar technologies can create problems trying to minimize the amount of OS noise generated.

The scheduler used, as well as pinning processes to specific CPU cores, could greatly impact the amount of OS noise being generated [14]. This project will try to avoid generating OS noise by reducing the amount of background tasks running while testing and running the load generator on a different physical CPU (where applicable) than the data gathering software.

When talking about virtualization, OS noise takes on an added dimension. The virtualization technology used could introduce an additional layer of scheduling and resource management. This layer can contribute to OS noise, which influences the performance and power consumption of the workloads.

What this project aims to measure is the differences in OS noise generated by the virtualization technologies tested.

## 4.8 RAPL - Running Average Power Limit

Intel's Running Average Power Limit (RAPL) is a hardware feature that allows monitoring of energy consumption across different domains of modern CPU chips. RAPL is considered an energy *estimation* feature that follow the overall power trends of the CPU. Not all readings are perfectly in line with actual power draw, most RAPL readings are off by a constant [11]. The power readings are closest to true power usage when DRAM (CPU cache) is being heavily utilized and any integrated graphics are not utilized. There is a linear correlation between RAPL package readings and full system power, with a correlation coefficient of 0.99. This is more accurate than high class smart energy meters [16].

RAPL supports getting energy measurements from different domains of the CPU chip. The domains supported differs between CPU models. Generally, RAPL supports the following domains:

- Package (PGK) - The entire socket. All CPU cores including their cache, integrated graphics and all shared memory between cores. Systems with multiple physical CPUs will have one Package domain for each CPU.
- Power Plane 0 (PP0) - All cores on the socket.
- Power Plane 1 (PP1) - Only the integrated GPU, if one exists.
- DRAM - This is the level of cache that is shared by all CPU cores.
- Psys - This was introduced when Intel released Skylake (2015) and is the entire system On a Chip (SoC). Not all versions of Skylake support Psys.

Getting RAPL readings on Linux is done programmatically through the Powercap API, which updates the running energy consumption continuously. Getting energy usage requires two readings with a fixed amount of time between each reading. The granularity of *sleep()* on Linux is very fine, which enables consistent readings from the API [9].

For this project, reading data from the Package domain is sufficient, since this gives the overall power consumption of the specific CPU socket. This will include the dynamic power consumption of the complete CPU.

## 4.9 Dynamic Power Consumption

When looking at the power consumption of a CPU, there are several factors contributing to its power usage. These are; the dynamic power consumption, the short-circuit consumption and the power loss due to transistor leakage. The CPU power consumption can be summarized as [4]:

$$P_{CPU} = P_{dyn} + P_{sc} + P_{leakage} \quad (1)$$

The dynamic power consumption ( $P_{dyn}$ ) is given by:

$$P_{dyn} = P_t(C_l * V * F)$$

$$P_{dyn} = P_t * C_l * V * F$$

where  $P_t$  is the probability that a power consuming event occurs,  $C_l$  is the loading capacitance of the circuit,  $V$  the supply voltage and  $F$  the frequency of the CPU.

The short-circuit consumption ( $P_{sc}$ ) is due to a direct path from supply voltage to ground. This happens when both parts of a CMOS transistor are activated simultaneously.

The last part of the power dissipation of a CPU is the transistor leakage,  $P_{leakage}$ . This effect is primarily determined by the manufacturing process and specification.

In a well-designed CPU, the dominant term of Equation 1 is the dynamic power consumption [4].

The effectiveness of a CPU comes down to a power to performance tradeoff. Increasing the frequency of a CPU increases both power consumption and performance. The same goes the other way, decreasing the frequency decreases both power consumption and performance. The specifics of these tradeoffs has been examined further in [12].

Moving beyond the CPU, it is imperative to understand that other hardware components also contribute to dynamic power consumption. Input/Output (I/O) operations, which communicate between local devices and external operations such as disk read and write activities, significantly impact the power consumption of a system. Network related I/O operations, especially the transmission of network packets through the internet, is known as one of the most costly types of operations, in terms of power consumption. Multiple factors influence the power consumption of each packet transmission, some of these factors include; distance, hardware specifications and the overall infrastructure on the packet's route. In 2022, the estimated energy usage of the data transmission network was between 260-280 TWh [1].

#### 4.9.1 Dynamic Voltage and Frequency Scaling (DVFS)

Voltage and frequency both play a major part in the power consumption of a CPU, with voltage being the major contributor [12]. Dynamic Voltage and Frequency Scaling (DVFS) is an Intel technology used to reduce the power consumption of a CPU. It does this by dynamically altering the frequency and the voltage of the CPU.

Manually controlling CPU voltage and frequency in a server environment can reduce power consumption at the cost of an equally significant amount of performance [12]. For this project, however, the main point of interest is not DVFS.

Understanding what drives the major factors in a CPU's power consumption is needed for creating a test environment that is fair. Keeping voltage and frequency constant through testing is a requirement for a good testing environment, we already know that the power consumption of a CPU changes with voltage and frequency [12].

## 4.10 Security

Security in a cloud environment needs to be applied at all software stack levels; from the application layer down to the hardware used. This section will focus on the differences between how containers and Virtual Machines help make the cloud environment a secure place. Both technologies need the same kind of active application layer security to secure the applications used. They do, however, differ in some ways when it comes to how the technologies make sure that an attacker can not break out of the virtual environment and, in the event of an attack, how the damage can be minimized.

*Isolation* is a technique used to ensure that an attacker can not break out of the virtual environment. Containers and virtual machines achieve isolation through different means. Isolation in a container environment is done through the host Operating System. In the case of Linux, this is done through *cgroups*, or Control Groups. A single *cgroup* is a collection of processes that are bound to a set of limits or parameters defined via the *cgroup* file system [3]. Windows uses a similar technique to achieve the same level of isolation. How secure the *cgroups* are comes down to the level of isolation achieved with the specific version of the Linux or Windows kernel used.

Virtual Machines achieve isolation through the hypervisor used. A hypervisor achieves isolation by giving the guest operating system virtualized physical hardware. A guest operating system is not aware of any other operating systems present under a hypervisor. As with Containers, the isolation level achieved depends on the piece of software or hardware creating the virtual environment. If the hypervisor is secure, isolation should be achieved.

Side-channel Attacks (SCA) is a concern in both single and multi tenant cloud environments. Some of the most prominent *isolation* violations comes from side-channel attacks. These types of attacks aim at exploiting a leaky channel to obtain sensitive data such as encryption keys. These attacks can be categorized in the following way; Timing attacks, Cache attacks, Electromagnetic attacks and Power-monitoring attacks. The Electromagnetic attacks and Power-monitoring attacks usually require physical access to the device, while Timing attacks and Cache attacks are the main software attacks [2]. Side-channel attacks can be completely avoided by not sharing any physical resources between applications. However, that could completely break the nature of having a cloud environment.

The general differences in how containers and virtual machines are used also changes how security can be applied to the technologies. Since containers are generally seen as short-lived instances with the ability to be restarted extremely

fast, they open up a few possibilities regarding security measures. If an attacker has managed to gain control over a container, it can be quickly and easily restarted. Restarting the container disconnects any open connection to the outside world and minimized the amount of time an attacker has to get out of the container and gain access to other parts of the system. Restarting containers could be done in a rolling manner for critical applications without downtime if replication is used with in the container Management Software. The same theory could be applied to virtual machines, but containers small, fast and agile nature makes them a better candidate for this type of security measure.

Several tools for automatically scanning container images for vulnerabilities exists. The tools accept a container image as input and uses a known vulnerability database to find any known vulnerabilities inside the container image. A study performed in 2021 [10] shows that these tools are not sufficient for any organization to rely on. The most reliable scanning tool found in the study missed 34% of all available vulnerabilities. The tools show better results when looking for configuration related vulnerabilities in the container themselves, but do not suffice for scanning nonOS related code inside the containers (application level code).

Configuration drift is the concept of how long-lived instances might have a different configuration than when they were instantiated. This could potentially become a problem when moving from a safe, known, configuration at instantiation to a configuration where more services than what is needed are exposed. Generally, this is more of a problem with virtual machines, since they have a longer lifespan. Problems associated with configuration drift can be completely avoided using a management software for either the containers or virtual machines used.

The most common attack vector for applications are vulnerabilities in outdated packages. 99% of exploited vulnerabilities are compromised more than a year after the CVE was published [7]. Being able to quickly and effortlessly update running applications packages is key to staying away from the large-scale automated attacks. Both a container and a virtual machine environment has support for this.

Container images are, generally, more standardized than virtual machine images in the sense that they contain less overhead, which makes them easier to automatically scan for vulnerable packages before release. Automatic scanning is available for virtual machines as well, but as size and complexity increases, so does the complexity of finding vulnerabilities.

The microservice architecture is often associated with containerized applications. The general design philosophy that comes with microservices mean that the network complexity could rise significantly compared to monolithic applications. Compared to a monolithic application where there might be one network path into the machine and the same network path out, microservices also require communication between themselves as well as communication with the Internet. This means that the network complexity will increase and as a result keeping all the open ports secure from outside attacks will be a bigger challenge in a microservice architecture. This does not imply that there is a security concern

with either containers or the microservice architecture, but a bigger challenge from a DevOps perspective keeping the attack surface to a minimum.

## 5 Testing Methodology

This section describes how the tests have been conducted.

### 5.1 Methods

The tests together with the data points gathered are used to design a model for the energy usage of the different environments. The end result is to give an idea of the power consumption differences in different types of applications.

Since all applications behave differently, the tests are designed to expose the underlying deployment technologies for different types of application scenarios. The following parts are evaluated:

- CPU heavy application
- CPU heavy application that uses `sycalls`
- I/O heavy application

All three parts that will be evaluated will be used to answer research questions 1 and 2 found in section 3.

During testing, multiple variables will be analyzed. Together with power consumption, each test will also gather data on CPU temperature and frequency. This is to be able to find if any tests create an environment where the CPU is not operating at its default clock rates, affecting power consumption through DVFS.

The following technologies will be evaluated:

- Bare metal - this will be a baseline for the power consumption of the application without any virtualization
- Docker - Container technologies generally leverage the same kernel level features exposed by the host operating system. Docker is the world leading container technology and is therefore used
- Virtual Machine - Hosted with a Type 1 hypervisor through KVM. Disk management is done through QEMU/libvirt

To simulate varying CPU workloads, the load generator will be able to stress different numbers of CPU cores, specifically utilizing 2, 4, and 6 cores while keeping the remaining cores idle. In a 6 core CPU, this translates to 33%, 66% and 100% CPU utilization.

## 5.2 Testing Hardware

This section explains the two hardware

### 5.2.1 Orbit

The tests are performed on a system which in this report will be referred to as Orbit. Orbit has the following system specifications:

- 2 x Intel® Xeon® Processor E5-2620 v2. Each CPU has 6 physical cores with hyperthreading. Hyperthreading was disabled during testing. It has a base frequency of 2.1GHz with a Turbo Boost frequency of 2.5GHz.
- 64GB DDR3 memory
- Solid State Drive for storage
- Ubuntu 22.04

The power consumption of the system was monitored using an HP/HPE Intelligent Modular Power Distribution Unit. It has the ability to monitor power consumption of connected units and can be accessed programmatically. These units update their power readings every 0.5 seconds with a granularity of 1 W.

Since each Orbit machine has access to two physical CPUs, and therefore also two RAPL domains to read from, it enables isolating the test environment. One of the CPUs is used to perform testing, while the other one is used for data gathering. This should in theory reduce the OS noise generated in the testing environment and subsequently in the results.

### 5.2.2 Consumer Grade CPU

With tools such as DVFS that try to minimize the power consumption of a CPU, it could be interesting looking at the difference between server and consumer grade hardware. There might be differences in both the hardware and software implementations of these technologies, since the use case for consumer and server grade equipment is very different.

The same tests were therefore also conducted on a consumer grade PC with the following hardware:

- Intel i5 9600k. The CPU has 6 cores. It has a base frequency of 3.7 GHz and a Turbo Boost frequency of 4.6 GHz.
- 16GB DDR4 memory
- A graphics card connected to the PC

- Solid State Drive storage
- Ubuntu 22.10

## 5.3 Tools

The following sections will describe the tools and software used during testing.

### 5.3.1 Data Gathering

Data gathering is done on the host of the virtualization technology used. It is done through the Linux Powercap API that reads values from RAPL, explained in Section 4.8. A Python script was developed that automatically runs a test suite with user specified time intervals and iterations for data gathering.

Data is collected with 1 seconds intervals following the start of the data gathering script. The tests gather data for 30 seconds in 10 iterations, giving a total of 300 data points for each test. This is to be able to distinguish between outliers in the test results.

### 5.3.2 Workload Generator

The workload generator used is a modified version of the open source application [Stress](#). It is a workload generator designed to subject a system to a configurable amount of workload through creating child processes and is written in C. *Stress* allows users to put a workload on different parts of the system, from simple CPU loads to I/O heavy operations.

Originally, the *Stress* application uses the following CPU workload loop:

```
while 1:
    sqrt(rand())
```

which is a great way to put a heavy load on a CPU. However, this proved fatal since the libraries used to perform these calls *could* be using `syscalls` and early test results indicated that this was probably the case.

To avoid having to deal with libraries, the application was modified to only operate on function local variables, completely avoiding calling any libraries. The following pseudocode is used for the CPU tests:

```
for i = 0 To test_lenght
    x = y * z
```

When compiling the application, the compiler flag `-O0` had to be used. Without it, the `gcc` compiler understands that none of the variables in the loop are ever used, and will therefore remove the loop when optimizing the code.

Testing a CPU heavy application while using `syscalls` is done by adding memory allocations onto the CPU test described above. The memory allocations are done through `mmap()` and `munmap()`. The size of the memory allocated increases with each loop iteration to avoid the operating system reusing the same memory for future allocations. Further, early tests showed strange behaviour when performing these tests. This was because, without ever using the memory asked for by a kernel, the memory might not even be allocated to the process. The memory is used by having `memset()` touch all the allocated memory, forcing the kernel to actually allocate it.

I/O consists of multiple types of operations, not all of them are tested. The workload generator is built to test I/O operations by creating a file, writing to that file, and then removing it from the file system. There is no way to do this without invoking `syscalls`. It does it in the following way:

```
for i = 0 To test_length
    open()
    write()
    close()
```

where writing is done in chunks of bytes.

Since the workload generator creates work for the machine by creating child processes, the amount of CPU usage in each test case depends on the amount of available CPU cores compared to the amount of child processes created. In our case, with both test benches having 6 core CPUs, each child process would contribute to 1/6 of the CPUs total available time.

## 5.4 Pinning Processes

Since Orbit consists of two physical CPUs, the testing takes advantage of this to minimize the amount of OS noise generated. This is done by only allowing the load generator in all environments used to run on one physical CPU and having the data gathering run on the other physical CPU.

The way this is achieved differs between the different deployment technologies used. Pinning processes during the bare metal testing were done using `numactl`, a software used to control which CPU cores an application is allowed to use. Docker as a built-in feature when running a container to specify which CPUs to use. Setting CPU affinity inside the virtual machine used is done through configuration changes through `virsh`.

## 5.5 Throughput

Throughput testing was initially done by doing calculations while the test suite was running. Each second of the tests were running, the load generator would interrupt to print relevant throughput data to the terminal. This was thought to be enough, since each second of performing testing should yield similar results due to the workload being the same through each test.

There is a huge problem with printing data during testing, each time the CPU has to print data, the testing procedure has to leave its current objective to print information to the screen. The tests are designed to show the difference between an application that uses `syscalls` and one that does not. Invoking `printf()`, which is a `syscall`, during testing could, and most likely did, affect the test results.

The throughput tests are therefore designed in a way where they keep track of the time before the test was started and when the test was completed. A known number of iterations were performed and could be divided by the total time taken for the iterations to be completed. These tests had to be redone to avoid the issues discussed above. Each technology tested might not share the same throughput, which means that a test on bare metal might not run for the same amount of time as a test on a virtual machine. This is not a problem as long as the tests have enough time to stabilize performance. Each test runs for at least 200 seconds, with some exceptions where the test needs more time to get through a good amount of iterations done. While doing throughput tests, 2 CPU cores will be kept busy. The results are reported as per core iterations per second.

The results from the throughput testing together with the results from power consumption testing will be used to answer research question 3 found in section 3.

## 5.6 Technologies

All the systems used in the testing are running Linux as host operating system. The hypervisor used was a Type 1 hypervisor. This was accomplished with `KVM`. It supports full virtualization for Linux on x86 hardware containing either Intel VT or AMD-V. KVM can be classified as both a Type 1 and Type 2 hypervisor because KVM creates private virtualized hardware for each individual machine while still keeping the host operating system usable. It has the performance of a Type 1 hypervisor with the ease of use of a Type 2 hypervisor.

The virtual machine used in testing is running Ubuntu `Minimal` which is a stripped down version of Ubuntu designed for automated deployment at scale. Using a minimal installation of any operating system should decrease the amount of measurable OS noise in the tests compared to a larger, more feature rich version of the same operating system. To facilitate creation, allocation and management of virtual machines `libvirt` was used together with `virsh` which is a CLI tool to manage virtual machines created with `libvirt`.

The container technology used in testing was **Docker**. With the similarities in how different container technologies work, there should be no difference in the CPU bound tasks. There could, however, be a difference in tests where I/O is used due to how different container technologies deal with virtualized storage.

## 6 Results

All the individual tests results from the different environments and on different hardware will be found in Appendix 8. Each test result displayed there contain the following information:

- The raw data points gathered in a box plot. This is to be able to distinguish between data that has too many outliers to be analyzed further. It's also a good visual representation of the data gathered, since OS noise will be very visible.
- The CPU temperature and frequency. During early testing, odd data points were identified to be the result of varying CPU frequency. CPU temperature could impact frequency, it is therefore also displayed.
- The overall energy usage of the complete system when performing tests on Orbit.

The raw data gathered is not included in this report. This is because of looking at the raw data without any processing does not make sense.

Where test data is presented using box plots, each box contains readings from one test iteration. One test iteration contains 30 data points.

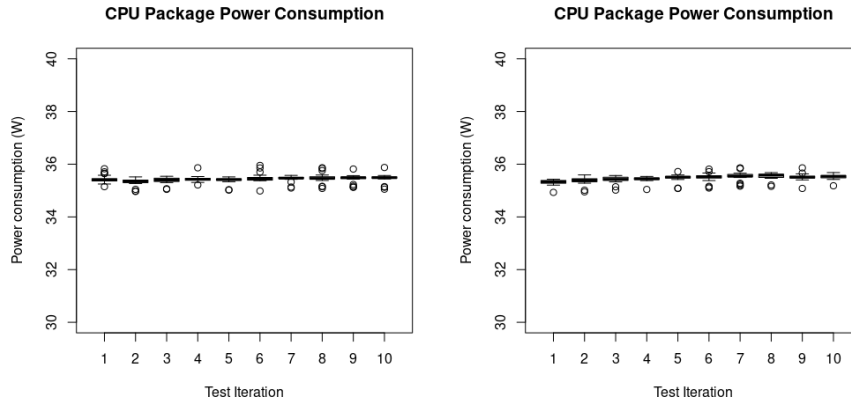
The amount of OS noise generated will be quantified using *Coefficient of Variance* (CV). It is a way to measure how spread out values in a data set are relative to its mean value. Since the data gathered does not share a common mean value (except in the case of equal power consumption), CV enables comparing the spread between the data anyway. It will be represented as a percentage and can be calculated as follows:

$$CV = \frac{\sigma}{\mu} * 100$$

This number will express the differences in the maximum and minimum values observed compared to the mean of a data set. A higher value implies that there is a larger variance in the data, indicating a higher amount of OS noise.

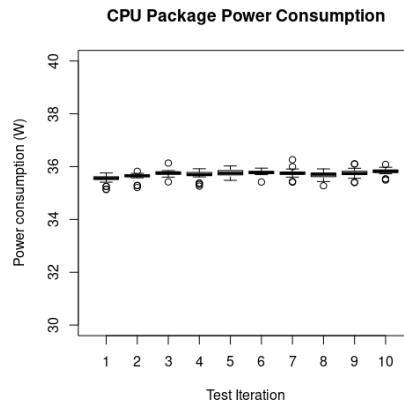
### 6.1 Orbit - General results

The tests performed on Orbit show a very stable testing environment when looking at CPU power consumption, frequency and temperature. Figure 8 shows the power consumption of the different environments tested with 4 CPU cores busy.



(a) CPU power consumption - bare metal.

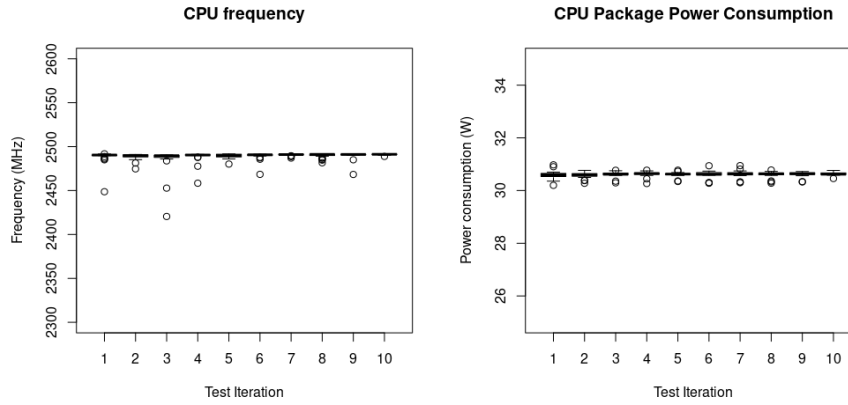
(b) CPU power consumption - container.



(c) CPU power consumption - virtual machine.

**Figure 2:** Stable power consumption readings. Data from CPU load test with 4 CPU cores busy.

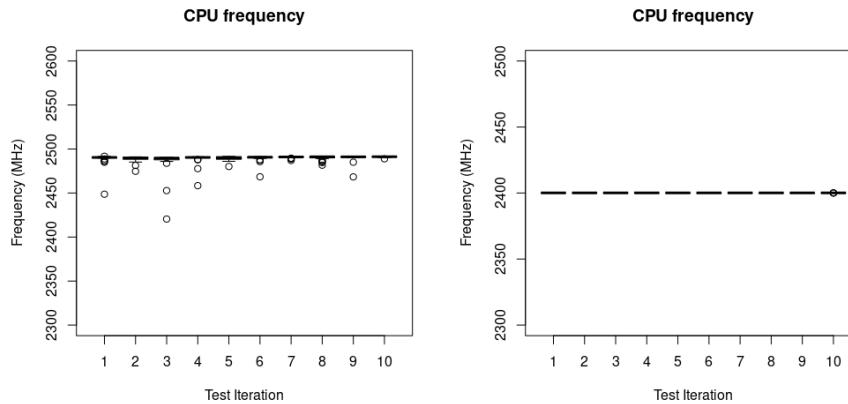
Fluctuating frequency could impact power readings. Throughout testing, the frequency of the CPU used did fluctuate in some readings. With a base frequency of 2.4GHz, a couple of test iterations had the CPU in Turbo Boost mode. This can be seen in Figure 3(a) but we can still see stable power consumption readings for the same test (Figure 3(b)). Power consumption readings stay stable even though CPU frequency is fluctuating, which indicates that there is not enough frequency fluctuation to impact the results.



(a) CPU frequency with load on 2 CPU cores using container. (b) CPU power consumption with load on 2 CPU cores using container.

**Figure 3:** Stable power consumption readings even though the CPU frequency was fluctuating.

The fluctuating frequency was only observed in tests with 2 CPU cores busy. With both 4 and 6 CPU cores busy, the CPU was very stable at 2.4GHz through the whole test. This effect can be seen in Figure 4.

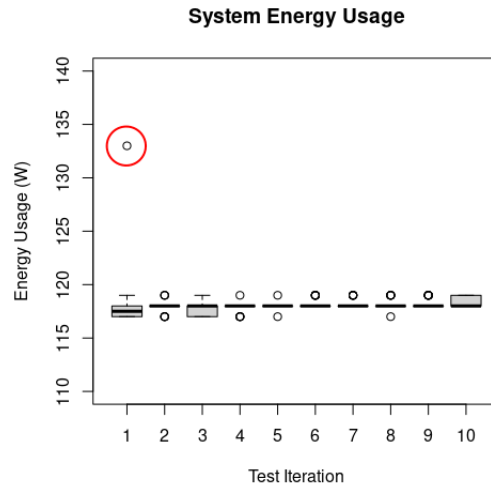


(a) CPU frequency with 2 CPU cores busy. (b) CPU frequency with 4 CPU cores busy.

**Figure 4:** Fluctuating frequencies only observed in tests where 2 CPU cores were busy.

The total power consumption of the system was also monitored during testing, together with the CPU power consumption. These results include the power consumption of all components used in the system. With Orbit, this means that the power consumption of both physical CPUs is included. Some of the tests performed show a spike in power consumption during the first test iteration. Figure 5 shows this power spike when performing CPU tests. This only happens in some cases, and it seems to be more prevalent when the data gathering software has not been started for a period of time. The spike in power

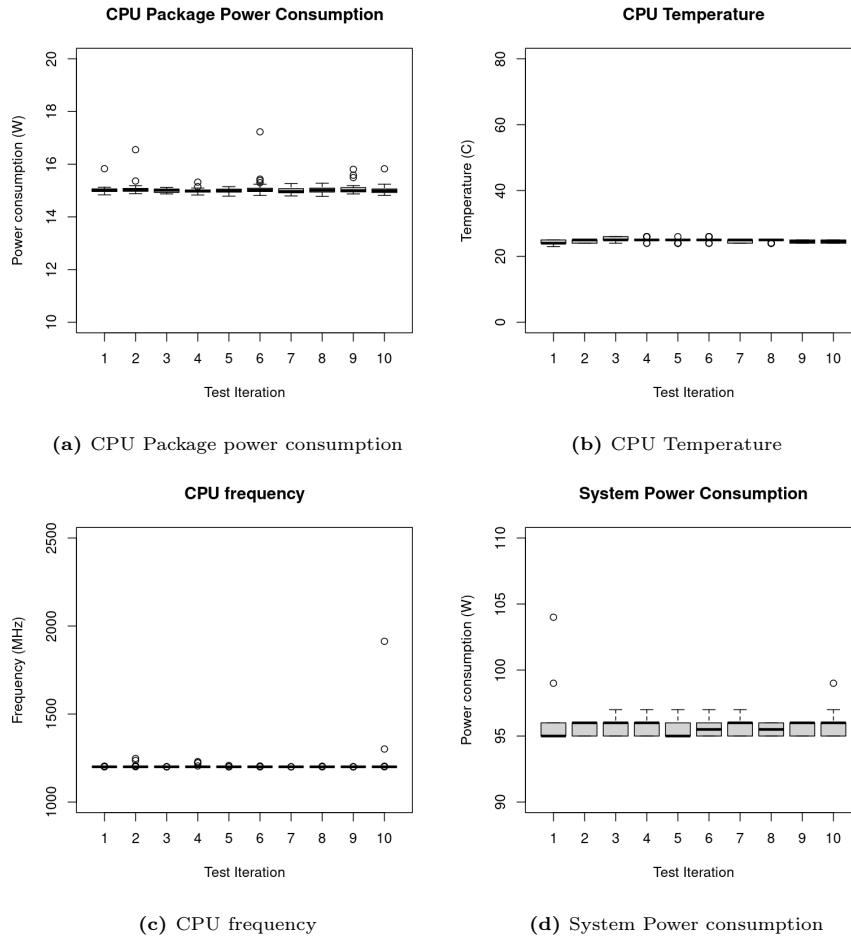
consumption is most likely an effect of the Python data gathering script loading libraries and resources it needs to be able to do its data collection. For analysis of the data, this data point can be removed. For future projects, this could be avoided by letting the Python data gathering script load its data and then sleep for a short period of time to let the power consumption stabilize before gathering data.



**Figure 5:** Power spike during gathering of complete system power consumption. The red circle marks the first power reading where the spike occurs.

### 6.1.1 Power Consumption Baseline

A power consumption baseline was created where power consumption data was collected without any specific workload present. This data will reflect the idle power consumption of Orbit.



**Figure 6:** Data from Orbit while the CPU is not under any synthetic load. This is the general power consumption of the hardware while the CPU is idle. System readings will include the load the data gathering software puts on the system.

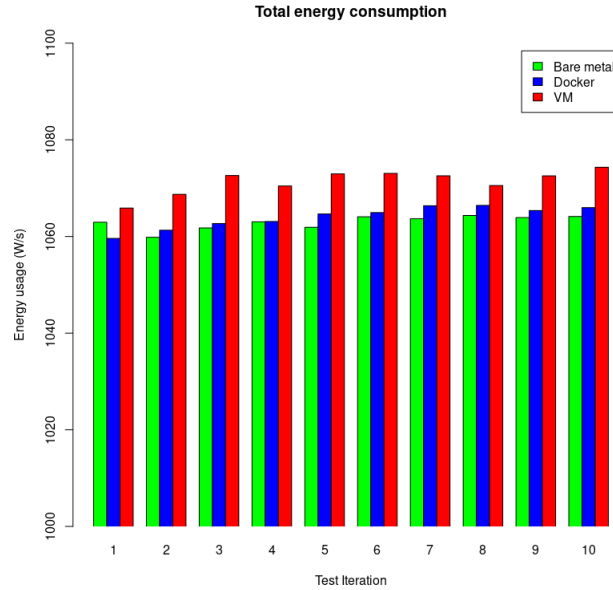
The mean CPU package power reading while Orbit was idle was 15.02 W. From Figure 6(a) we can see that the power consumption is not completely stable while the system is not under any load. But, while the power consumption is not completely stable, the frequency of the CPU (Figure 6(c)) did stay mostly constant throughout the test, with only a couple of readings being outside the interquartile range. With this in mind, we can conclude that the CPU can use more power without increasing its operating frequency.

### 6.1.2 CPU Tests

This section will discuss the results related to the CPU tests described in section 5.3.2.

Total energy consumption for each test iteration using 4 CPU cores can be found

in Figure 7. It can be seen that a virtual machine uses more energy than both bare metal and a container.



**Figure 7:** Total energy consumption for each test iteration when performing CPU heavy workload without system calls. Load generation on 4 CPU cores.

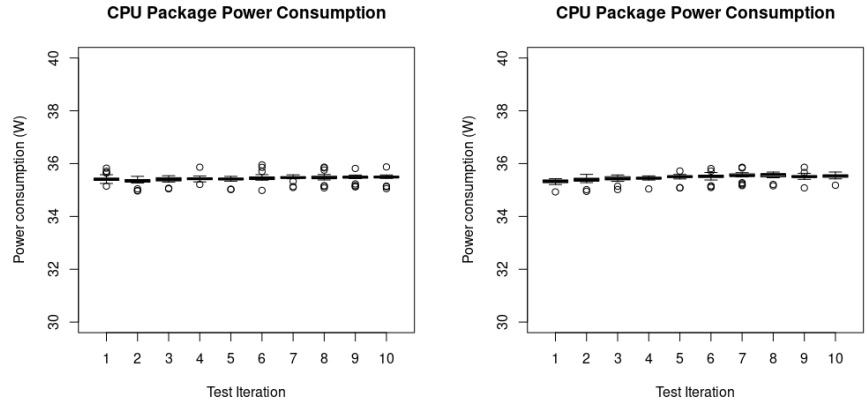
Table 3 shows the mean power reading and the total energy consumption for the same test as Figure 7, 4 CPU cores. The difference in energy consumed between the lowest consuming technology (bare metal) and the highest (virtual machine) for all 10 test iterations is 84.03W/s. Since each test is performed during 300 seconds, the difference if the tests were performed during 24 hours would be, 24200.64W/s of energy. This equates to a 0.0067 kWh energy consumption increase using a virtual machine compared to bare metal for a full days' operation.

The difference in energy consumption between a container and a virtual machine can also be found in Table 3. The difference in total energy consumption is 73.21W/s, which an increase of 0.6%. During one day of operation, the difference would equate to 21084.48W/s of energy savings using a container technology instead of a virtual machine. Converted to kWh, this is 0.0058568 kWh of energy saved operating a full day on a single physical CPU. Remember that these numbers only reflect a single machine operating. In a cloud data center, the number of physical machines could be huge.

	Mean power reading (W)	Total energy consumption (W/s)
Bare metal	35.43208	10629.62
Container	35.46812	10640.44
Virtual Machine	35.71218	10713.65

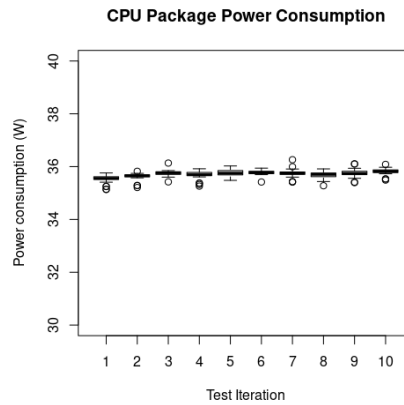
**Table 3:** Results from CPU tests running on 4 CPU cores.

Figure 8 contains power readings from each technology tested using a load on 4 CPU cores. The mean value and the variance in those readings can be found in Table 4. Any OS noise encountered during testing will be found in the variance of the power readings. Since the mean value changes for each test performed and in each environment, looking at only mean values and variances makes it hard to distinguish which environment experiences the most amount of OS noise.



(a) CPU power consumption - bare metal.

(b) CPU power consumption - container.



(c) CPU power consumption - virtual machine.

**Figure 8:** Visualization of OS noise with load on 4 CPU cores within all three testing environments.

It is not clear whether there is a difference in the amount of OS noise generated in testing by looking at Figure 8. In the testing data, the OS noise can be quantified by looking at the variance in each data set. Table 4 contains the mean values and variances from each environment tested. Quantifying the amount of OS noise generated is hard by looking at the data found in Table 4. If OS noise was generated at a fixed percentage, say 10%, that means that a higher power consumption will lead to more OS noise generated. None of the environments share a common mean value, which in turn means that the variances will not be directly comparable when looking for OS noise. Therefore, the Coefficient of

Variance can be used to make comparing the variables easier.

	Bare metal mean (var)	Container mean (var)	VM mean (var)
2 CPU cores	30.66 (0.013)	30.62 (0.011)	30.70 (0.015)
4 CPU cores	35.43 (0.018)	35.47 (0.019)	35.71 (0.024)
6 CPU cores	40.97 (0.039)	40.94 (0.028)	41.07 (0.022)

**Table 4:** Mean values (W) and variances (W) in data from tests. A lower variance to mean value ratio would indicate lower amounts of OS noise.

Table 5 contains the CV values for the test performed. It shows that there is a difference in OS noise between the environments, it is, however, not clear which environment generates the least amount of noise. Except for, in most cases, an increasing CV value with increasing core count, there is not a clear pattern in the data.

	Bare metal CV	Container CV	VM CV
2 CPU cores	37%	34%	39%
4 CPU cores	38%	39%	43%
6 CPU cores	48%	40%	36%

**Table 5:** Coefficient of Variation. Note that a CV value of 48% does not mean that there is a 48% variance in the data. The CV values show the spread of data, a higher value means the data has a higher variance in relation to its mean value.

So far, most of the discussion has been about tests performed while 4 CPU cores are busy. This situation has the closest resemblance to the average CPU utilization in data centers. To verify that the difference in energy consumed by the environments holds for 2 and 6 CPU cores, a Welch Two Sample t-test was used, and the results can be found in Table 6 where the data collected on containers and virtual machines can be found. The three t-tests conducted all show a p-value of  $< 0.05$  which can be interpreted as that the null hypothesis can be rejected. This means that there is a significant difference in the two data sets for all tests conducted on a 95% confidence interval.

	Container mean (W)	Virtual Machine mean (W)	p-value
2 CPU cores	30.61647	30.69633	$< 0.001$
4 CPU cores	35.46812	35.71218	$< 0.001$
6 CPU cores	40.93718	41.07438	$< 0.001$

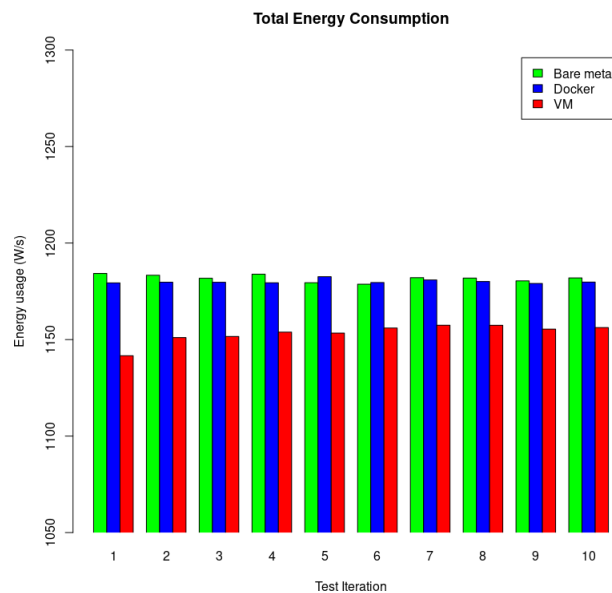
**Table 6:** Welch Two Sample t-test for all CPU tests performed on containers and virtual machines.

The power consumption increases with both virtual machines and containers, with the difference in power consumption of a virtual machine being 0.6% higher than a container. Depending on the scale of the data center, this might be something to keep in mind but when deciding on which virtualization technology to use. With how small the differences in power consumption is, this should probably not be the deciding factor when looking at containers and virtual machines.

### 6.1.3 CPU Tests with System Calls

This section will discuss the test results where the system has been stressed with CPU load together with `syscalls`. As with the previous section, most of the discussion will revolve around the test case where 4 CPU cores are utilized.

The total energy consumption of the tests can be found in Figure 9. The results are very different compared to the tests that were performed without any `syscalls` (Found in Figure 7) with the virtual machine using less energy than both the container and bare metal.



**Figure 9:** Total energy consumption for each test iteration when performing CPU heavy workload with system calls. Load generation on 4 CPU cores.

These results are unintuitive, since the extra virtualization layer between a virtual machine and bare metal should, in theory, increase energy consumption. In this case, an extra virtualization layer decreases the energy consumption.

As discussed in section 5.3.2, the `syscalls` being invoked during testing are `mmap()` and `munmap()`. These are very similar to `malloc()` and `free()` in the sense that they ask the operating system for memory to use and signal the operating system that the memory is no longer needed. KVM does guest memory allocations by having a user space program on the host asking the host kernel for memory through `malloc()` whenever memory is needed in the guest<sup>1</sup>, which is a form of caching. This means that there is another layer of abstraction between the guest calling `mmap()` and the host operating system actually committing memory to the process. The virtual machine used has 2GB of memory allocated at initialization. If this theory is correct, this means that

<sup>1</sup><https://www.linux-kvm.org/page/Memory>

those 2GB can be used as a memory cache for the guest operating system. If the guest operating system has not yet used the pre-allocated 2GB of memory, there is no need to let the `mmap()` call go through to the host. The exact reason behind the lower power consumption in a virtual machine with these specific tests needs to be investigated further, with more research to give a clear answer. Either more research into the virtualization layers or a thorough call tracking analysis could be used.

Using the test performed without `syscalls` and comparing them with `syscalls` can be used to analyze if there is a difference in power consumption in the different environments when the CPU load is different.

Table 7 contain the mean values of the power consumption in the different environments with and without `syscalls`. It is clear that the power consumption increased in all environments with `syscalls`.

	Bare metal mean	Container mean	Virtual Machine mean
Without <code>syscalls</code>	35.43208	35.46811	35.71217
With <code>syscalls</code>	39.39238	39.33413	38.85467

**Table 7:** Mean values for power consumption from tests performed with and without `syscalls`. Mean values measured in Watt.

The ratio of power consumption increase can be found in Table 8. Both bare metal and the container have a power consumption increase of 11% while the virtual machine saw an increase of 8%.

	Bare metal	Container	Virtual Machine
Ratio of increase	11%	11%	8%

**Table 8:** Ratio of increase in power consumption, comparing CPU load with and without `syscalls`.

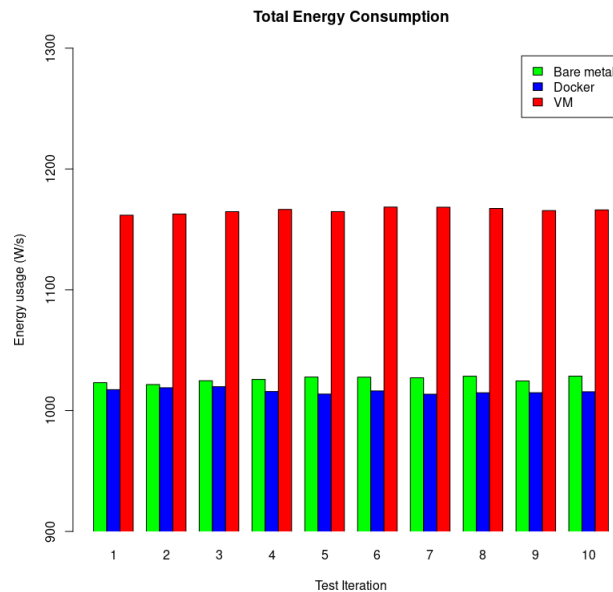
All three deployment environments tested show an increase in power consumption when performing `syscalls` while the CPU is busy. The virtual machine used in testing show a lower power consumption than bare metal and a container. Since the use of `syscalls` could be an integral part of an application, the power consumption increase is unavoidable. With how the virtual machine used in this project handles memory allocations through caching, these kinds of tests might show different results if they were performed on another type of virtual machine or if caching of memory was disabled.

Welch Two sample t-test were performed on data from the three different environments tested. All p-values observed were  $< 0.001$  which again means that the null hypothesis can be rejected, there is a significant difference between the power consumption of the different technologies.

### 6.1.4 CPU Tests with I/O

These test results are from running CPU tests with I/O calls included. The type of I/O called being invoked is discussed further in section 5.3.2.

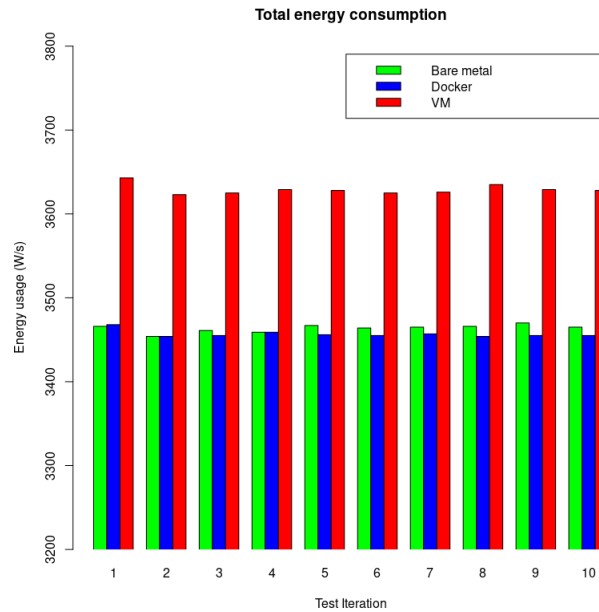
The total amount of energy consumed during testing with I/O calls can be found in Figure 10. This is the largest gap in energy consumption between the different environments experienced, where the virtual machines use a considerably larger amount of energy than any of the other technologies.



**Figure 10:** Total CPU package energy consumption for each test iteration when performing CPU heavy workload with I/O. Load generation on 4 CPU cores.

Since I/O operations, compared to CPU only tests cases, in theory should also increase the total power consumption of the system, the power readings for the complete system will be analyzed. These measurements were taken in conjunction with the rest of the data, but has not yet been analyzed since the workload has been very CPU focused thus far.

The total energy consumption of the complete system can be found in Figure 11 where it is, once again, clear that the virtual machine uses more energy than any other technology.



**Figure 11:** Total energy consumption of the whole system for each test iteration when performing CPU heavy workload with I/O. Load generation on 4 CPU cores.

An interesting aspect would be if the energy consumption of the complete system increases with I/O operations. Since I/O operations include other components than just the CPU, in theory, the energy consumption of the system should increase. Table 9 contain the energy consumption of the complete system with and without I/O operations. The results show that the energy consumption decreases with I/O operations for bare metal and containers, while the virtual machine sees an increase in power consumption with I/O operations. The lower power consumption in bare metal and containers can be explained by the fact that each I/O operation uses `sync()` to not buffer the `write()` operation. This means that the CPU could have to spend cycles waiting for the disk to perform the `write()` operation, the empty cycles will decrease power consumption of the CPU.

	Bare metal	Container	Virtual machine
Without I/O	35403	35422	35405
With I/O	34637	34568	36291
Percentage difference	-2.2 %	-2.4%	+ 2.5%

**Table 9:** Energy consumption for the complete test with and without I/O operations with 4 CPU cores busy. Energy consumption measured in W/s.

These results are from a system that uses a Solid State Drive (SSD) for storage. The type of storage media will most likely change the result of the tests. Using a SSD will increase the performance of the `write()` calls since they generally have

higher throughput than Hard Disk Drives (HDD) <sup>2</sup>. Drives with higher write speeds leads to the CPU having to spend less time waiting for the `write()` calls to complete, and should therefore lead to increased throughput and power consumption. Due to the internal structural differences between SSDs and HDDs, they do not share the same power consumption profiles. Some studies show that the active power consumption of SSDs are higher than that of HDDs <sup>3</sup>. Whether the type of disk used makes a difference requires more testing.

### 6.1.5 Throughput

With each test performed on Orbit, throughput was also monitored, results of which can be found in Table 10. With power consumption and throughput as a single unit, we can conclude which deployment environment gives the user the highest performance per Watt.

	Bare metal	Container	Virtual machine
Without syscalls	235457159	226840299	222136487
With syscalls	32621	30184	27008
With I/O	21043	10647	27842

**Table 10:** Results of the throughput tests performed on Orbit. All tests conducted with 2 CPU cores busy. Throughput is measured in iterations per second, and the results are average per core throughput.

When the test suite is only operating on function level variables (the test performed without `syscalls`) it is clear that both containers and virtual machines are approaching levels of throughput very close to that of running the application without any virtualization. The virtual machine is only 5% slower than bare metal at executing application level code while being only 2% slower than the container.

With the power consumption results when the application uses `syscalls` in mind where the virtual machine used less power than any of the other technologies, the throughput results might help support that claim. A lower throughput means that the CPU spends time waiting for data, this would mean that the CPU would not require as much power since it is not performing as many calculations per second.

When I/O is involved, the throughput of the virtual machine rises through the roof. Once again, the virtualization layers between the guest OS and the host could have caching involved when dealing with I/O which causes this behavior. Each `sync()` call in the guest OS performed by the test suite might not actually be written to physical disk, but instead a middle layer acting as a physical disk. To get accurate throughput results that line up with the theory behind virtualization layers, further understanding of how disk management is done in a hypervisor is needed together with further configuration of the virtual disk drives.

<sup>2</sup>SSD vs HDD performance

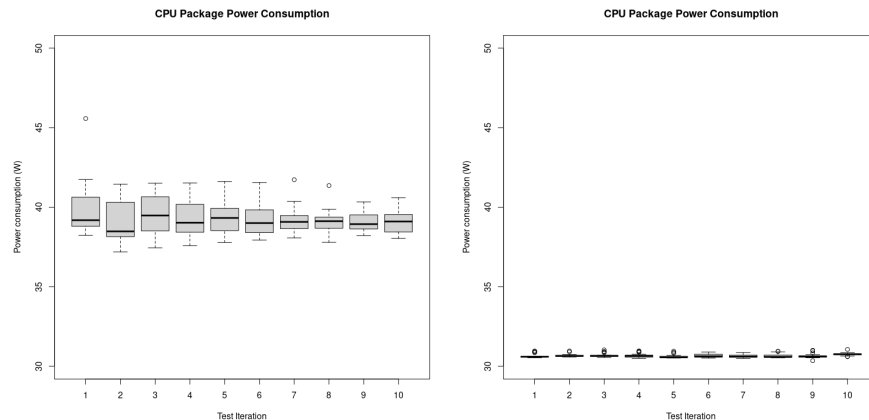
<sup>3</sup>SSD vs HDD power consumption

The container used (Docker) seems to handle I/O substantially worse than bare metal.

## 6.2 Tests Performed on Desktop PC

The following section will discuss the results gathered from a Desktop PC with desktop grade hardware and software running the base operating system. All the software and tools used in the different deployment technologies are the same as those used in testing on Orbit.

The results from the desktop PC show very large values of OS noise in the data compared to the results from Orbit. When comparing both test results found in Figure 12 we see that the server grade CPU in Orbit is consuming more power than the desktop PC. The desktop CPU power consumption fluctuates a lot more than the CPU in Orbit. The fact that the CPU uses more power in the desktop PC is because of the design of the CPU. The fluctuating power consumption in the desktop CPU can probably be explained by the design of the CPU as well. A server grade CPU and a desktop grade CPU are not designed to be used in the same kind of workloads. Where the desktop CPU is most likely designed to produce a system with low response times and power consumption, the server grade CPU is designed to perform continuous work.

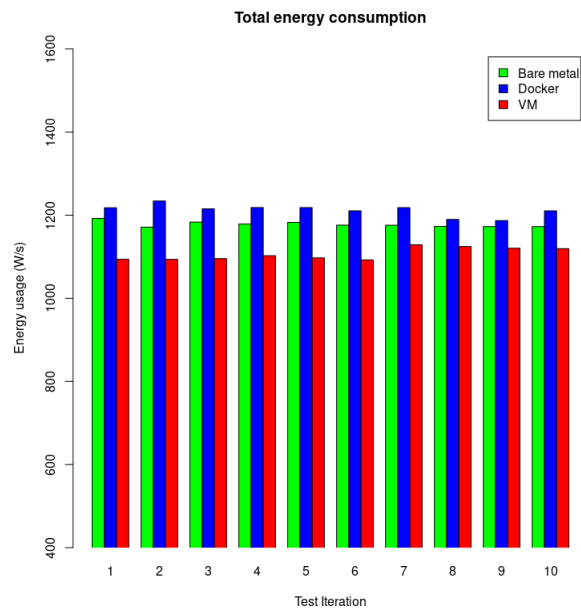


(a) Power consumption on desktop PC with 2 CPU cores busy. (b) Power consumption on Orbit bare metal with 2 CPU cores busy.

**Figure 12:** Differences in OS noise generated in desktop PC compared to Orbit.

### 6.2.1 CPU tests

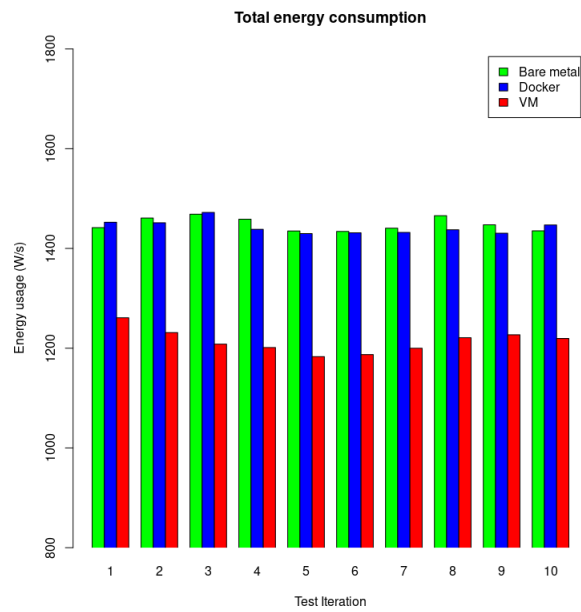
The CPU test performed on a desktop PC shows different results from those performed on Orbit. In Figure 13 it is clear that the virtual machine has a lower power consumption than bare metal and a container. This is a complete contradiction to the same test performed on Orbit (Figure 7) where the virtual machine has the highest power consumption.



**Figure 13:** Total energy consumption with all three deployment technologies on a Desktop grade PC with 2 CPU cores busy.

### 6.2.2 CPU tests with syscalls

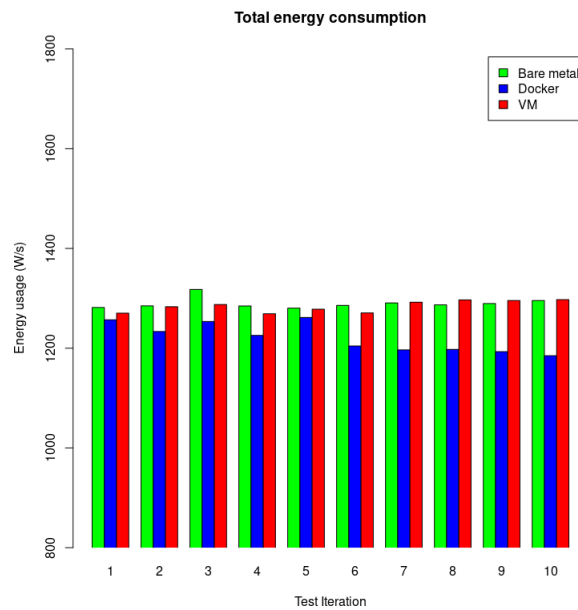
The tests performed on a desktop PC with `syscalls` show a similar trend as the CPU tests discussed in the last section. The virtual machine had a lower power consumption than any other deployment method tested. The test results can be found in Figure 14 which contain the total energy consumption for each test iteration.



**Figure 14:** Total energy consumption with all three deployment technologies on a Desktop grade PC with 2 CPU cores busy with `syscalls`.

### 6.2.3 CPU tests with I/O

Using I/O with a desktop PC lead to very similar test results for each of the deployment methods tested, where the container technology uses less energy than any of the other technologies used. The results of these tests can be found in Figure 15.



**Figure 15:** Total energy consumption with all three deployment technologies on a Desktop grade PC with 2 CPU cores busy with I/O.

#### 6.2.4 General Thoughts on Desktop PC

The desktop PC tested on was the system used in developing the data gathering software and all the test methods used in this project. It does, however, not mean that it is suited for this kind of testing. As seen in the results from Orbit, the variance of the data points is generally very small. A single odd data point could turn the results one way or another. It is therefore paramount for generating good data that the test environment is isolated. The desktop PC used does not create a well isolated environment for these kinds of measurements, since it contains a heap of features required for a desktop PC to function. These include a graphical user interface that needs to be updated, a graphics card that could require CPU time, multiple accessories connected over USB and generally an experience optimized for day to day usage.

This can be shown by looking at the CV value from the results of the desktop PC. Table 11 contains the coefficient of variation for CPU tests performed on Orbit and the desktop PC. The desktop PC has a way higher CV value for all tests performed. For these results, this means that the Desktop PC generates a substantially larger amount of OS noise.

	Bare metal CV	Container CV	VM CV
Orbit	37%	34%	39%
Desktop	265%	498%	362%

**Table 11:** Coefficient of Variation. A higher number indicates a higher spread in data readings, which in this case means more OS noise. Data gathered on Orbit and Desktop PC with 2 CPU cores busy on bare metal.

Since the OS noise generated is so high on the desktop PC compared to Orbit, the test results from the Desktop PC should probably not be used as hard evidence towards any difference in power consumption. The desktop PC fails to support an isolated test environment for these kinds of tests. Whether this is because of specific hardware or software features implemented in a desktop environment or the fact that Orbit has access to multiple CPUs for further isolation requires more testing.

### 6.3 Data Transfer Power Consumption

During the literature study for the project, further questions about power consumption of network heavy applications came to mind. After discussions with system administrators at Umeå University, a plan was developed to measure power consumption impact of network traffic in data center grade network equipment.

The following was tested:

- Does the type of network switch change the power consumption profile?
- Does sending data through network equipment change its power consumption?

#### 6.3.1 Hardware

The following hardware was used:

- 3x Dell PowerEdge R530 with a dual "Broadcom Limited NetXtreme II BCM57810 10 Gigabit Ethernet"
- HPE Intelligent Modular Power Distribution Unit, AF527A + AF547A
- MikroTik CCR2216-1G-12XS-2XQ Router (powerful CPU, for packet inspection) (12x 25Gbit, 2x 100Gbit)
- MikroTik CRS354-48G-4S+2Q+RM Switch (weak CPU, not for packet inspection) (48x 1Gbit, 4x 10Gbit, 2x 40Gbit)
- Connected with 1m SFP DAC cables.

### 6.3.2 Test Results

There was little to no difference in the energy consumption of either of the network switches tested when running in idle and performing high data switching and inspection. Table 12 contain the results of the testing performed.

	CCR2216-1G-12XS-2XQ	CRS354-48G-4S+2Q+RM
Idle	44W	16.5W
Network cables connected	45W	17W
Sending 30GB/s	46W	17.5W

**Table 12:** Power consumption of network switches while idle, with network cables connected and while sending data. Tests performed by Tomas Forsman (stric@cs.umu.se)

The energy consumption of the network switches used show more or less static power consumption. This means that sending data over copper wires through these network switches do not, in any meaningful way, impact the power consumption of the network equipment.

Fiber optics is used where either high bandwidth or longer transmission range is needed. This requires transceivers that convert the digital signal to light. One of these transceivers that can send data 100km has a maximum power consumption of 1.8W<sup>4</sup>.

The power consumption of a data communication network with this type of hardware is very static. More measurements could be done, but with these facts in mind, they would not contribute to the question this project aims to answer.

## 6.4 System on a Chip

As with the data transfer power consumption, another topic of interest that was thought of during discussions with supervisors through the initial literature study was if the power consumption of smaller, System on a Chip (SoC) computers would also have been affected by the virtualization environments. These computers could then have been set up in a cluster environment that somewhat simulates a data center cluster. The SoC that was thought to be used were Raspberry Pie computers.

There were a couple of problems with testing with Raspberry Pies:

- The Raspberry Pie computers used about 5W idle and 7W while doing computations. This would probably be measurable, but compared to larger CPUs the difference between idle and busy is very small.
- Since the difference between idle and busy is very small, the final analysis of the data would not be very interesting looking at.

<sup>4</sup>[www.fs.com](http://www.fs.com)

Measurements on these types of systems could be done, the problem lies with the very small, 2W, difference between having them idle and busy. While it is measurable, it is a very small difference in power consumption compared to larger CPUs.

## 7 Final Thoughts

This project set out to answer these research questions:

- RQ1. How does the power consumption of containers compare to that of virtual machines?
- RQ2. Does the type of application change the power consumption characteristics?
- RQ3. What are the implications of considering a more power efficient virtualization solution in terms of application throughput?

### 7.1 RQ1

This project has shown that in CPU bound environments, the power consumption of containers is lower than that of virtual machines by a small but statistically significant amount. The testing showed that virtual machines use about 0.6% more power than a container performing the exact same task when 4 CPU cores are utilized.

### 7.2 RQ2

The type of application drastically change the power consumption characteristics of the virtualization environment used. There is a clear difference in the power consumption of each virtualization environment when the application uses `syscalls` for memory management, where the virtual machine used in testing had a lower power consumption than both the container and the application running on bare metal. This behavior is unexpected. This project explains this discrepancy in power consumption with the virtual machines' virtualization technology having a cache of memory, meaning that the `syscall` does not need to go through the host operating system with each subsequent call.

Applications that use heavy I/O operations also change the power consumption characteristics drastically, with virtual machines having a higher power consumption than containers when I/O is involved. While containers 2.5% less power with I/O than without, virtual machines use about 2.5% more energy when the application running is a I/O heavy workload.

### 7.3 RQ3

Application throughput measurements require heavy knowledge of the underlying virtualization layers to create an environment where the tests performed are considered fair. Without any `syscalls` or I/O involved in testing, the throughput of containers and virtual machines are very close (Table 10), the virtualization layers used in this project only adds a small overhead in terms of CPU processing. With this in mind, we can say that containers are more efficient than virtual machines when performing CPU bound tasks.

With the test suite performing `syscalls`, the container has a 11% higher throughput. This is however offset by a small margin since the container used about 1% more power during testing. With throughput and power consumption as a metric, the container is more efficient at performing `syscalls` than the virtual machine is.

When testing the application with I/O involved, the test results are not very intuitive. The virtual machine has a 13% higher throughput than bare metal with I/O, which should not, in theory, be the case. With the knowledge gained during this project, I/O testing requires a high level of knowledge of the virtualization layers to create a fair testing environment. Adding a virtualization layer should decrease performance, not increase it, this makes it very difficult to take the results found at hand.

### 7.4 General Final Thoughts

Testing for specific behavior in an application requires a good understanding of the whole software stack, from the operating system down to the application layer. Without this understanding, it is very hard to gauge what exactly is being tested. This project required quite substantial reworks of the testing suite due to not understanding the whole software stack and how differently an application behaves when exposed to different kinds of `syscalls`. An operating system might not be as deterministic as it might seem at first glance. One might expect an operating system to perform in a very deterministic fashion since it, after all, is a computer running but without external tools, most of what is performed on an operating system follows some kind of scheduling policy, that most of the time is not exposed to the end user.

There is a great deal of difference in how data center grade hardware and consumer grade hardware works, which is understandable. The consumer grade hardware will probably experience very low average utilization and is therefore tuned towards that. Server grade hardware is supposed to perform work and is tuned towards that end goal. DVFS is a great example of a technology that is adapted depending on where the hardware is used, consumer grade hardware seems to be tuned towards keeping power consumption and heat to acceptable levels.

With these results in mind, it is very hard to recommend a single virtualization technology that is more power efficient than the other, as it depends heavily on

the type of CPU load imposed. Since the differences overall are quite small, the recommendation is to choose the virtualization technology that best suites the needs of the organization. Both containers and virtual machines comes with their own opportunities and weaknesses.

Other power consumption measures, such as holistic scheduling, has managed to reduce power consumption by as much as 10 – 20% [17] and should probably be considered before looking at a new virtualization technology if reducing power consumption is of great interest.

## 8 Future work

This project was set out to find the differences in power consumption between containers and virtual machines. These technologies generally require applications that are designed differently. With how containers generally fit together very nicely with a microservice architecture and virtual machines being more suited for monolithic applications, a holistic view on the problem might be better suited for the organization running the data center. A microservice application will most likely require more than one container running to fulfil its purpose. To give a full view of how these technologies can change the power consumption as a whole requires more testing.

This project could be extended in several ways. One very interesting aspect that could change the view of power consumption would be measuring each service used. Here, a microservice architecture could be compared to a monolithic application. There are open source projects that do this, one of them is [scaphandre](#) which permits power consumption measurements of individual services.

Isolation of the test environment is, as seen in the results of the Desktop PC, is very important for generating good data. Cold and hot starting the testing environments could generate different results for the first couple of tests performed. This has already been experienced with the oddities in the complete system power consumption discussed in section 6.1. Some of the data points are oddities and easy to find in box plots, smaller discrepancies in the data could still exist. The testing environment could probably still be isolated further, which increases the precision of the data.

There are more scenarios that could be of interest than the three that have been tested; without `syscalls`, with `syscalls` and with I/O. With the rise of cloud and microservices, application inherently become more network heavy, which was one of the reason for trying to include network testing. It would be interesting comparing how applications behave divided into services instead of applications. This would, instead of the focus of this project, be if, for example, a monolithic application has a higher or lower power consumption than a similar application with a microservice architecture.

The type of `syscalls` used in testing could greatly impact the amount of work being done by the kernel. This testing suite uses dynamic memory allocation as the invoked `syscall` which in itself should not be a very expensive operation

to complete. It could be interesting looking at how the type of `syscalls` would impact the results.

This testing used QEMUs standard way of managing disks, which is by creating a file on the filesystem that can then be mounted and interpreted as a disk drive inside the guest operating system <sup>5</sup>. Since the results of the I/O operations with virtual machines goes against both bare metal and container, where the virtual machine consumes more power while the other two use less power with I/O, investigating further with different types of ways a Hypervisor can handle disk management. For example, setting up a disk pass through for the guest operating system might lead to a higher throughput, and therefore, in the case of I/O testing, could reduce power consumption.

There are multiple different options to choose from when selecting a virtual machine host system and a container technology. It could have been interesting looking at the differences between different types of virtual machines and container technologies when investigating power consumption. Early testing showed that the difference between LXC and Docker containers was negligible and was therefore not investigated further, but after further research the difference might be noticeable in I/O operations.

---

<sup>5</sup><https://www.qemu.org/2020/09/14/qemu-storage-overview/>

## A Results

This Appendix will be dedicated to showing the data gathered from testing. The test results of interest have been discussed in earlier sections of this report, and this appendix only serves as a place where data that was gathered is displayed.

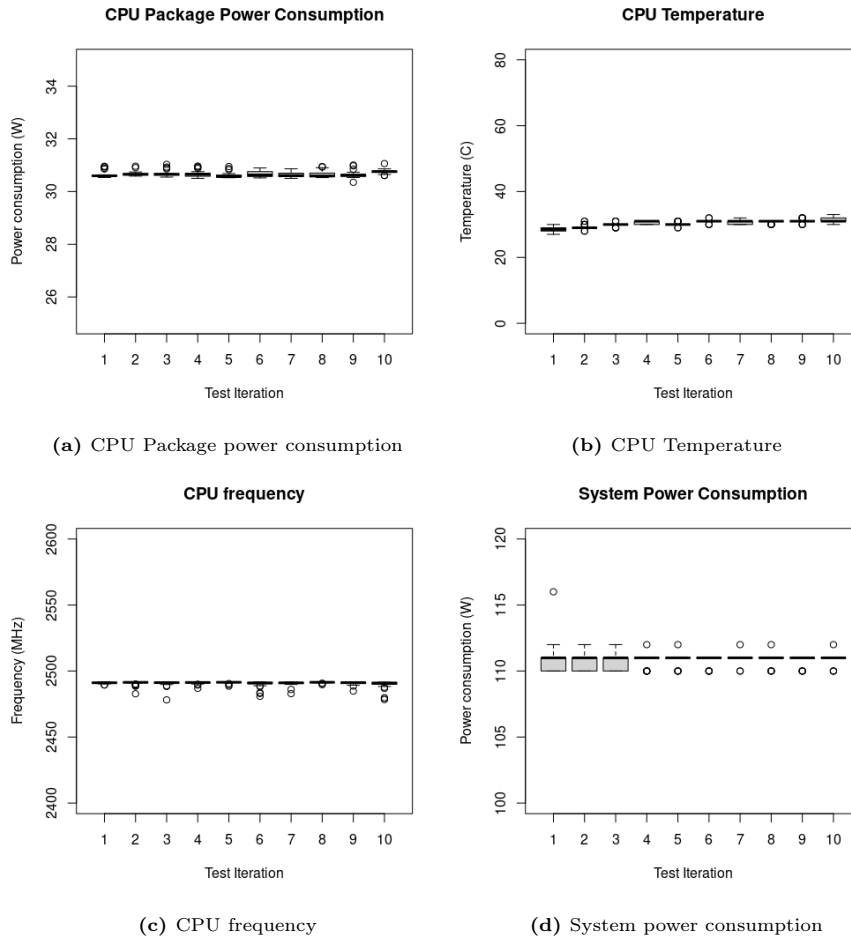
### A.1 Orbit

These tests results were gathered on Orbit.

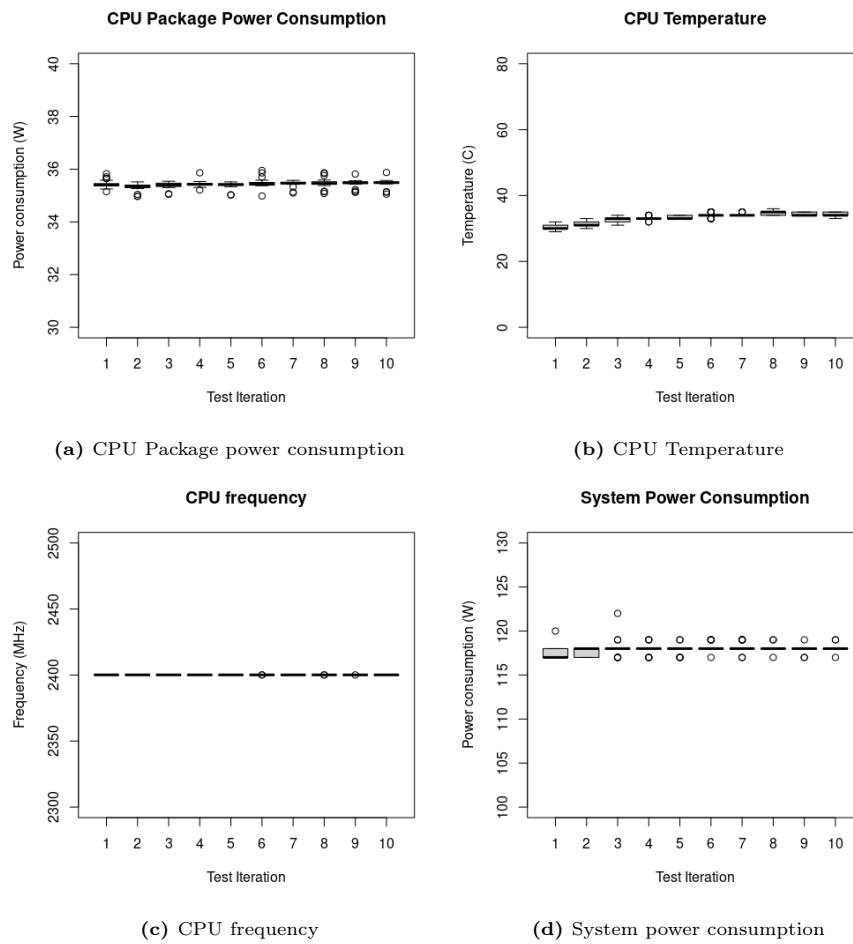
The section names indicate which test the results are from.

## A.2 Bare metal

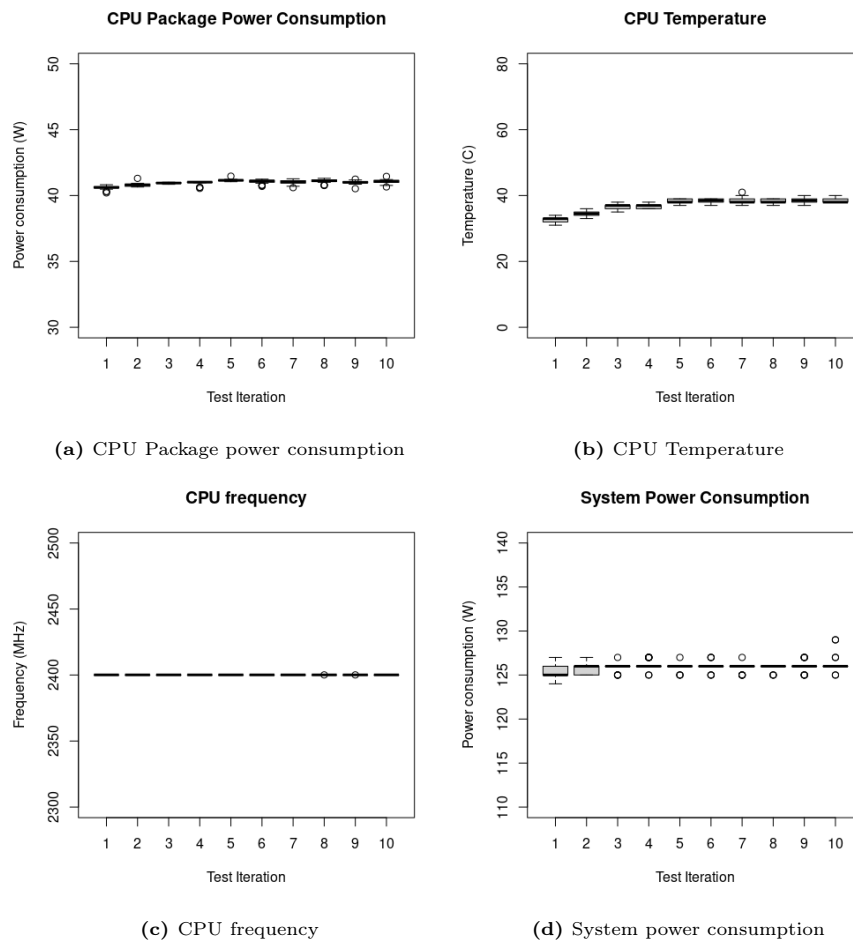
### A.2.1 CPU load without system calls



**Figure 16:** Test results running load generator on bare metal on 2 CPU cores.



**Figure 17:** Test results running load generator on bare metal on 4 CPU cores.



**Figure 18:** Test results running load generator on bare metal on 6 CPU cores.

## A.2.2 CPU load with system calls

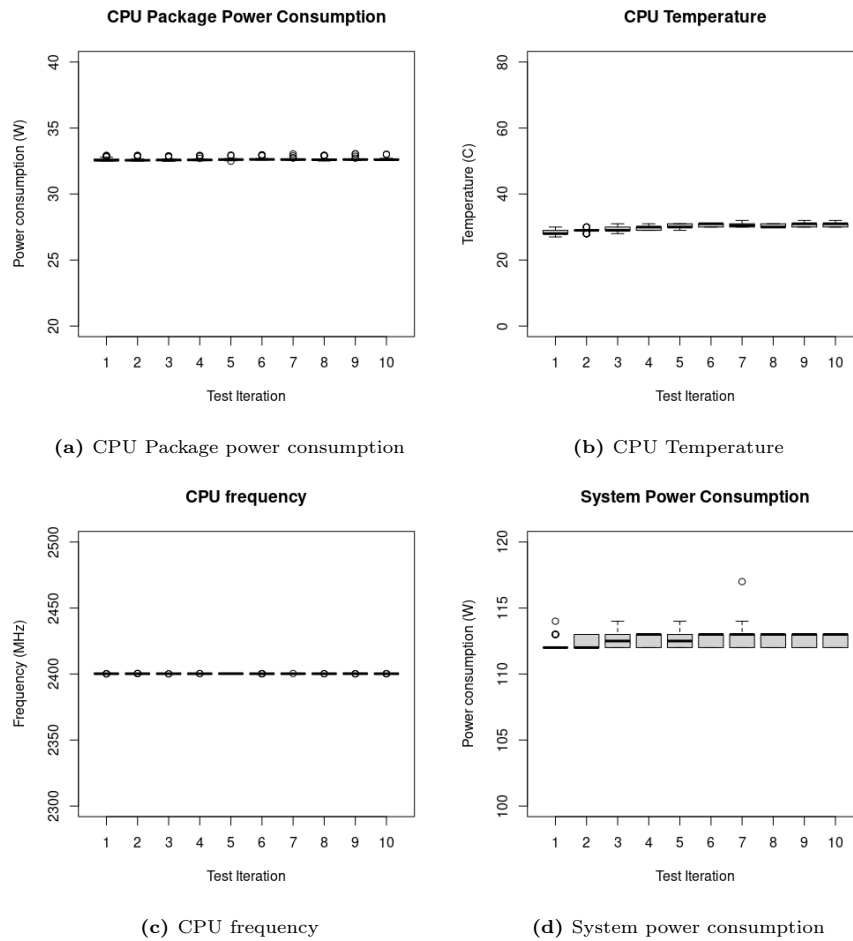
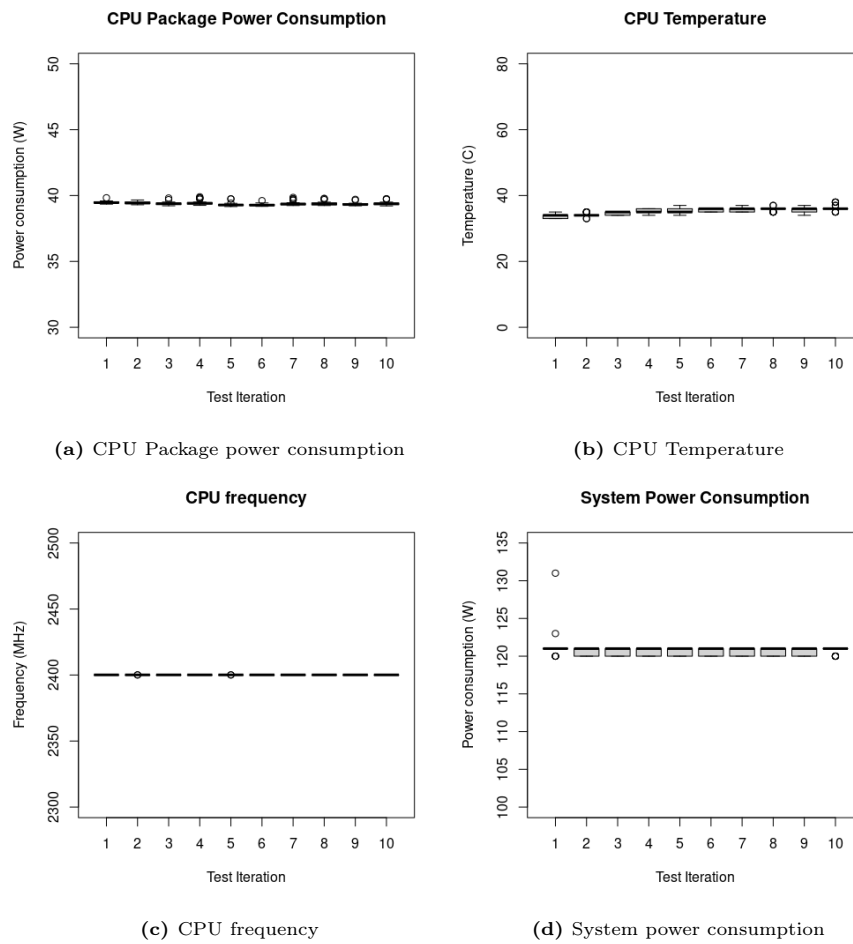
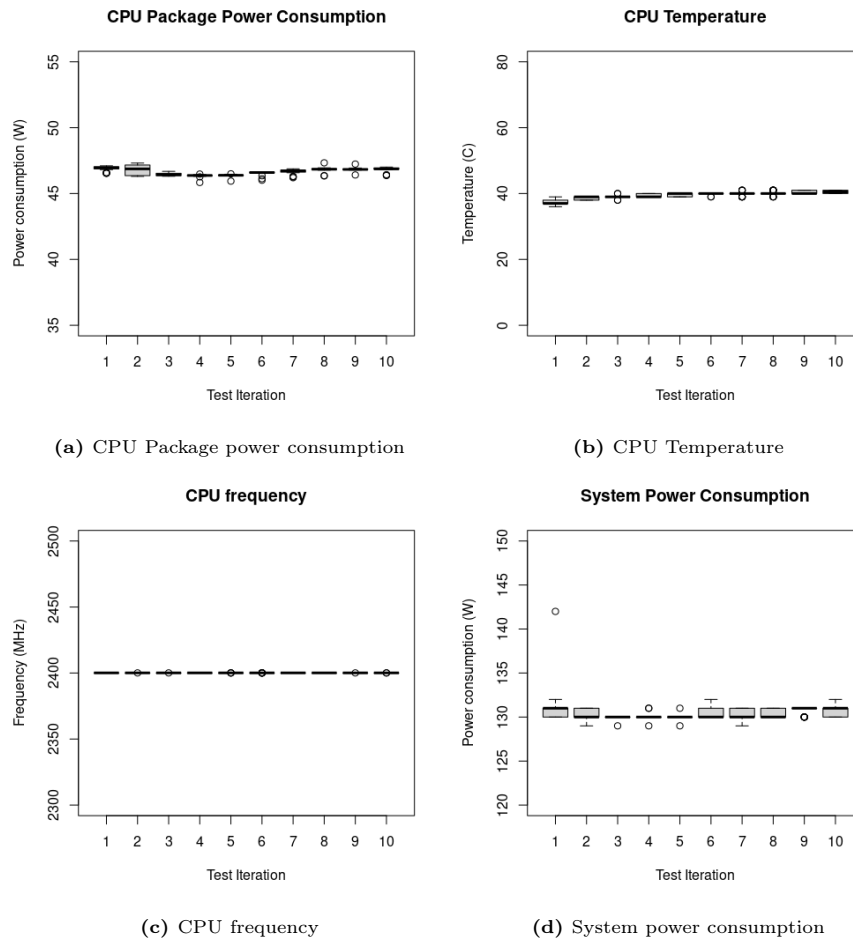


Figure 19: Test results running load generator on bare metal on 2 CPU cores.



**Figure 20:** Test results running load generator on bare metal on 4 CPU cores.



**Figure 21:** Test results running load generator on bare metal on 6 CPU cores.

## A.2.3 CPU load with I/O

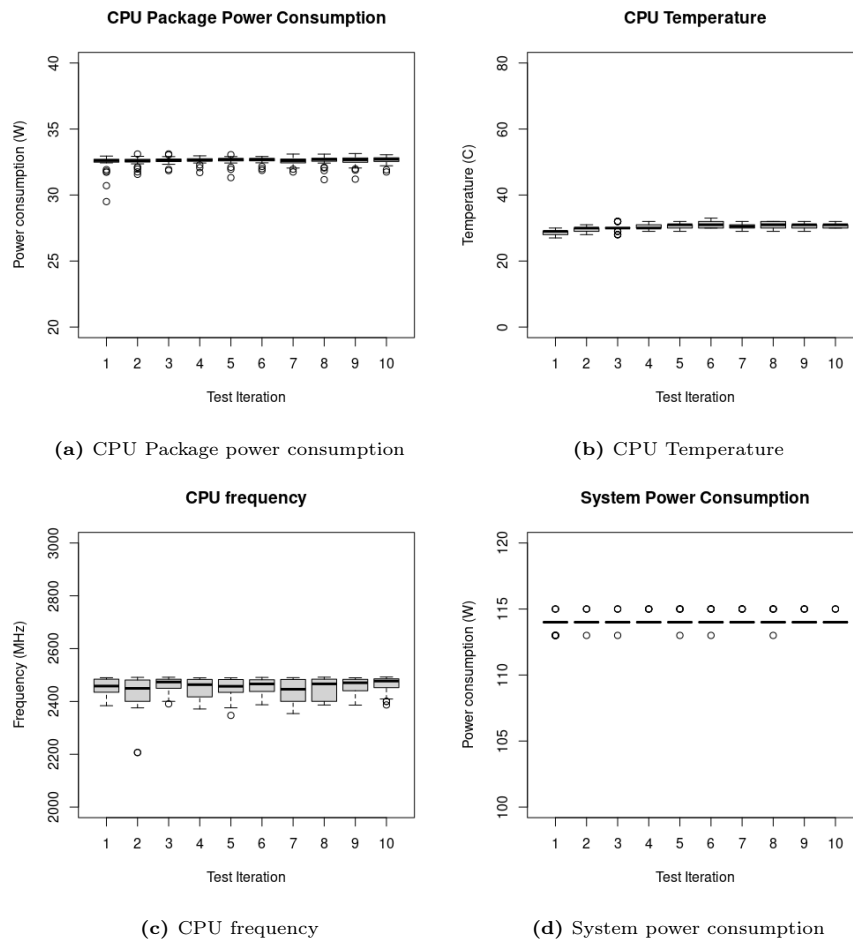
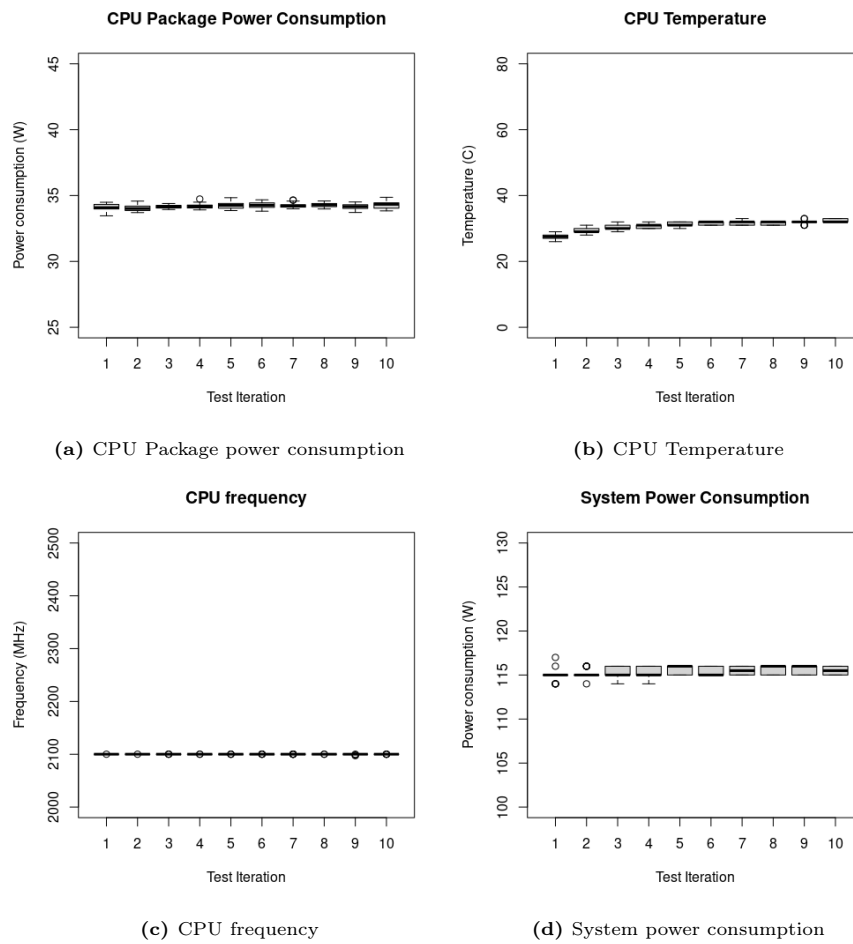
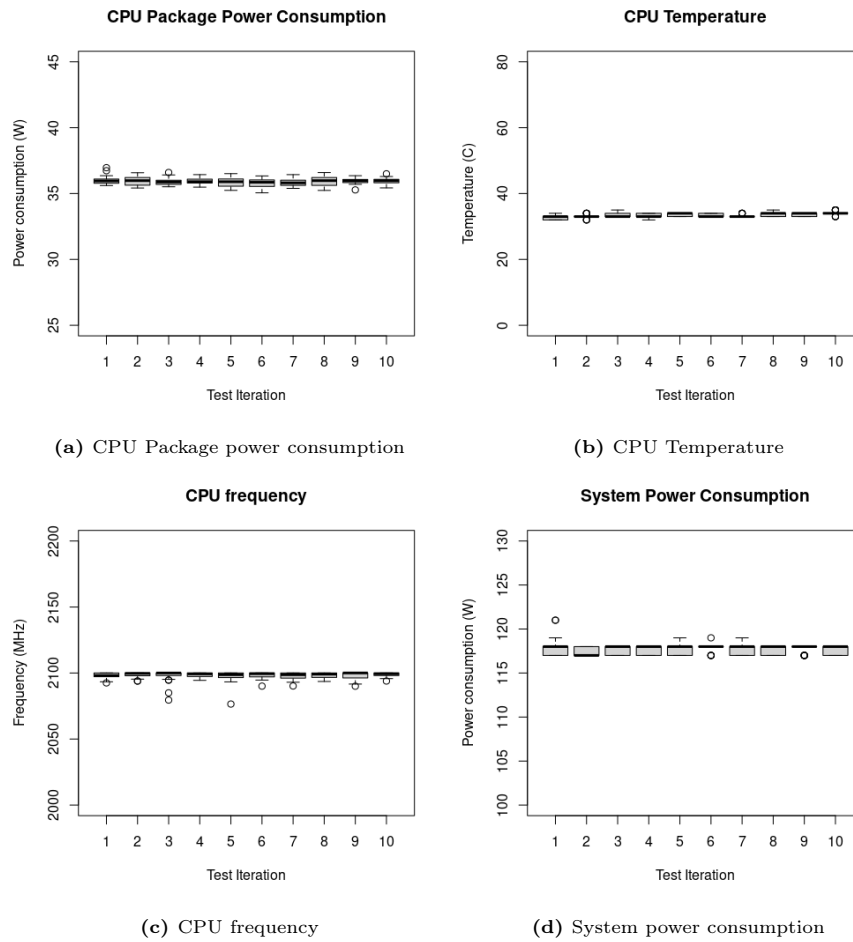


Figure 22: Test results running load generator on bare metal on 2 CPU cores.



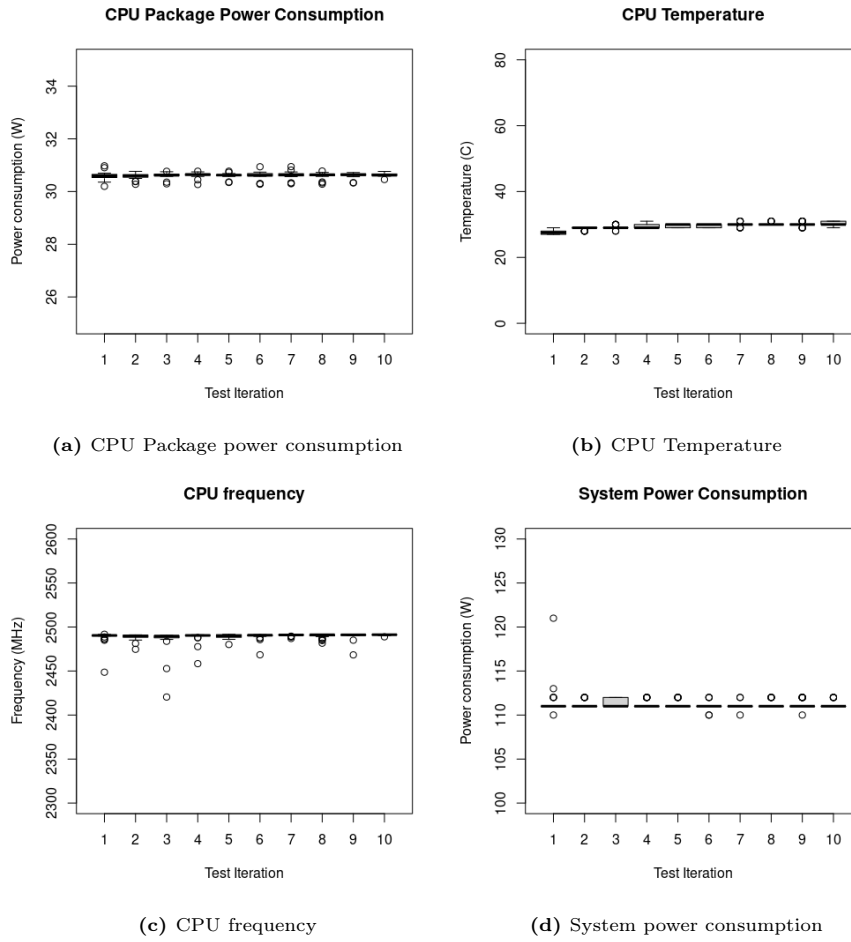
**Figure 23:** Test results running load generator on bare metal on 4 CPU cores.



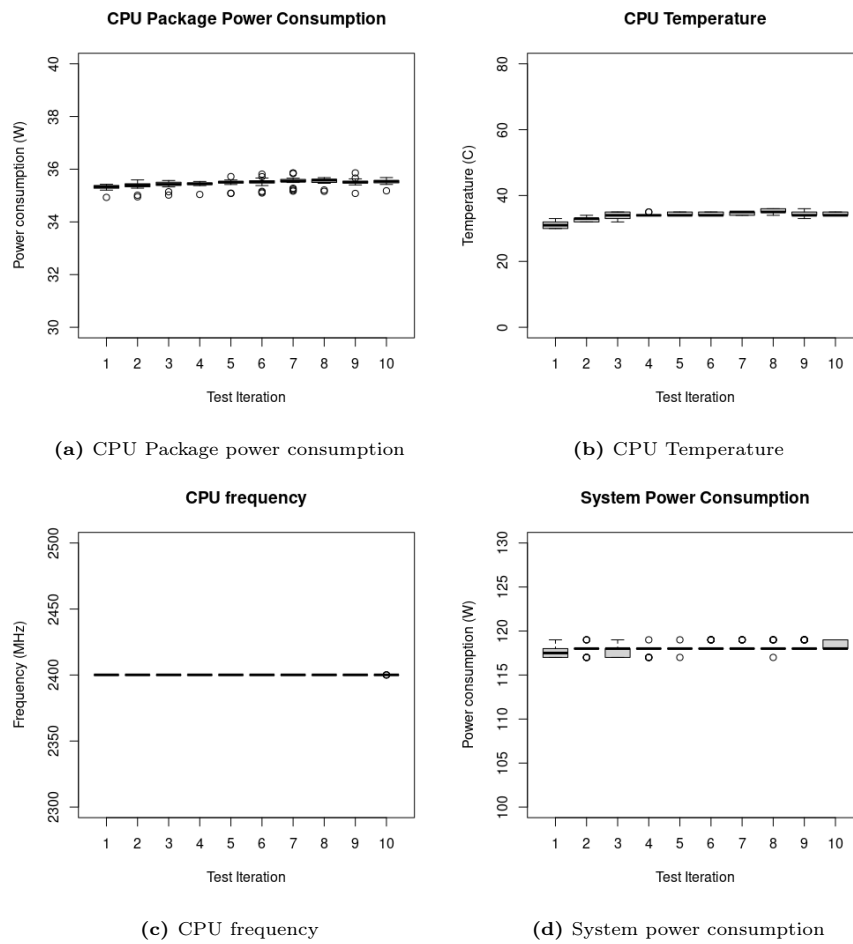
**Figure 24:** Test results running load generator on bare metal on 6 CPU cores.

## A.3 Container

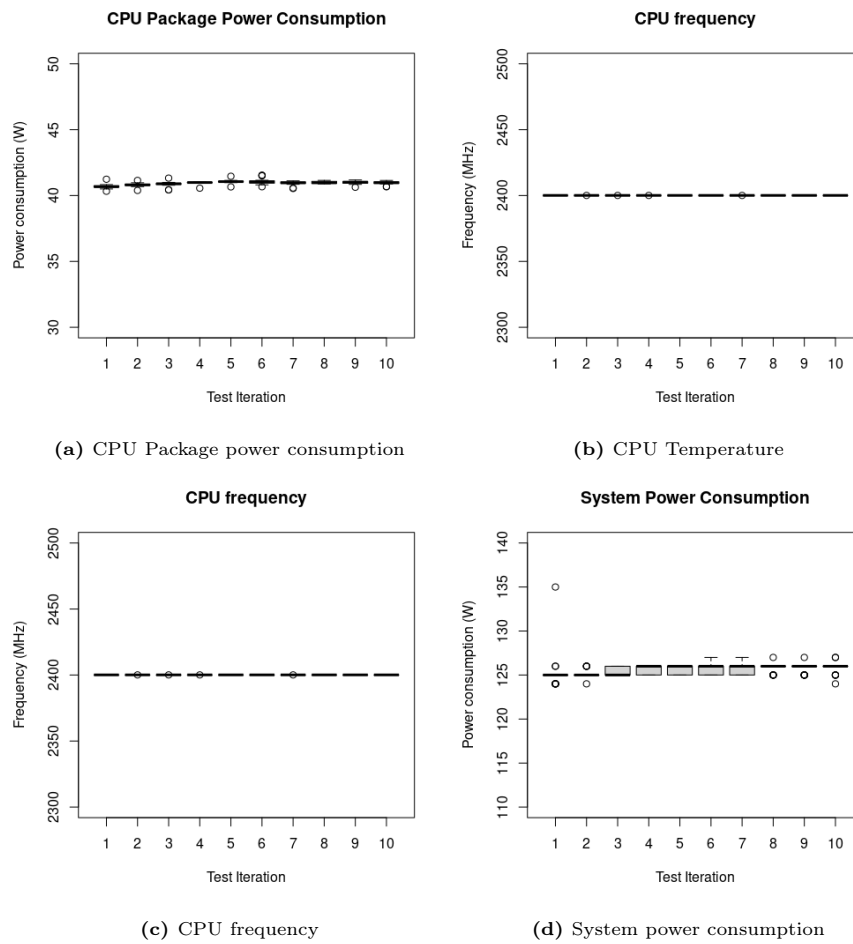
### A.3.1 CPU load without system calls



**Figure 25:** Test results running load generator on Docker on 2 CPU cores.

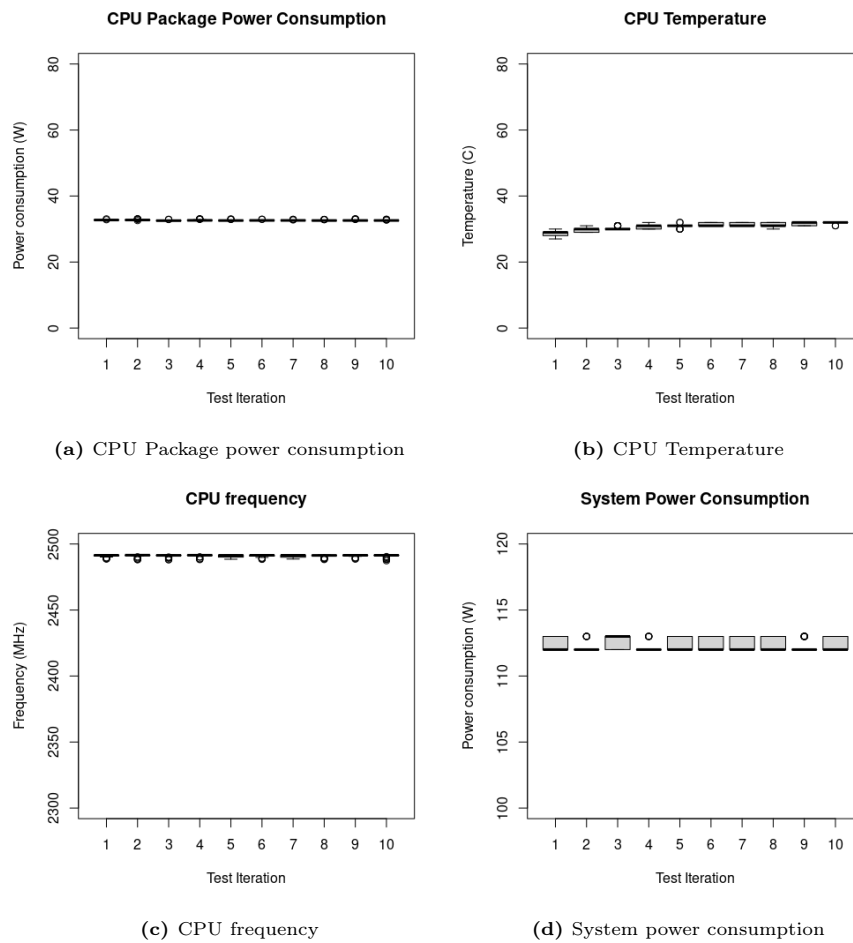


**Figure 26:** Test results running load generator on Docker on 4 CPU cores.

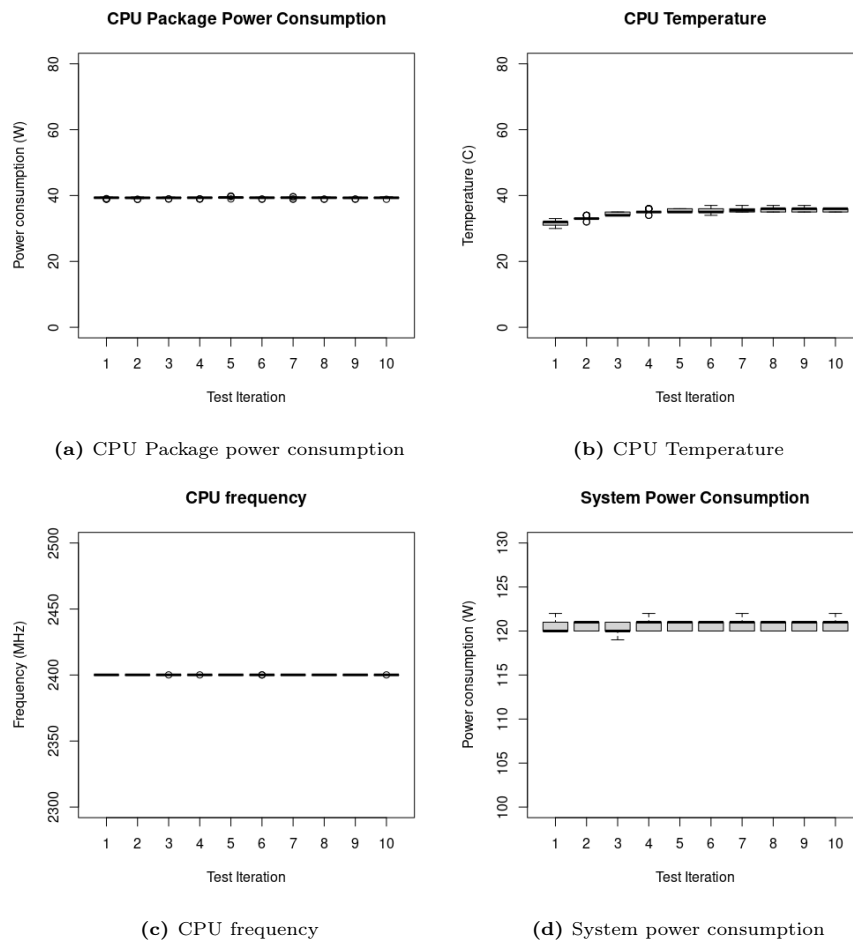


**Figure 27:** Test results running load generator on Docker on 6 CPU cores.

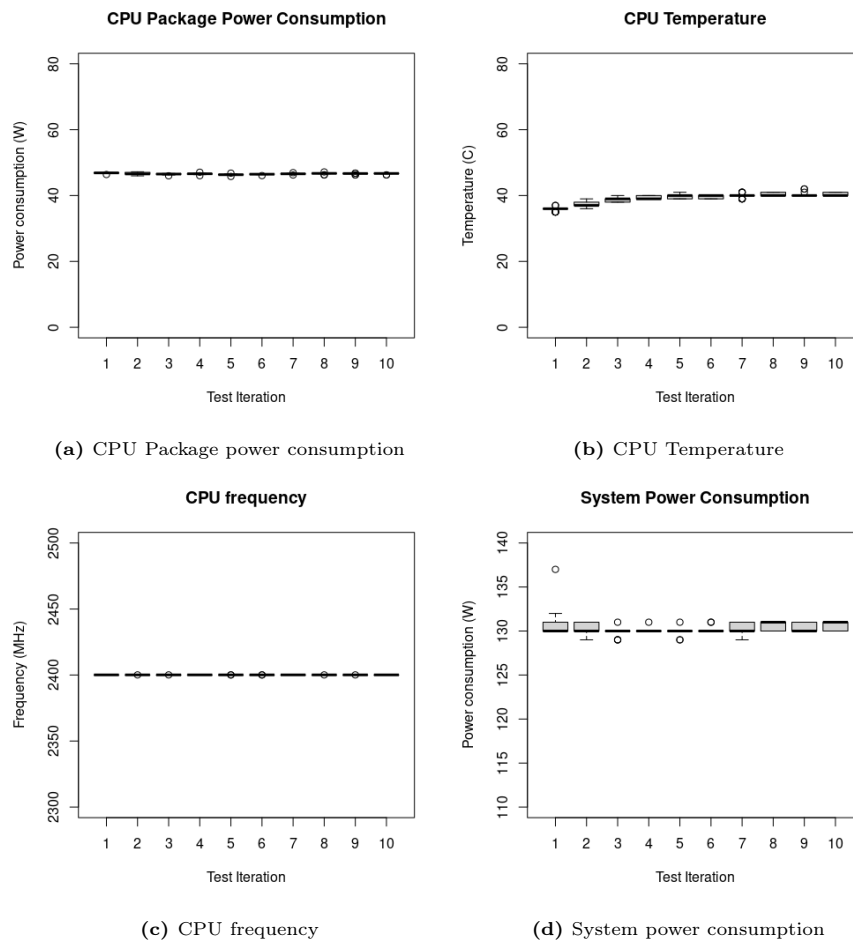
## A.3.2 CPU load with system calls



**Figure 28:** Test results running load generator on Docker on 2 CPU cores.

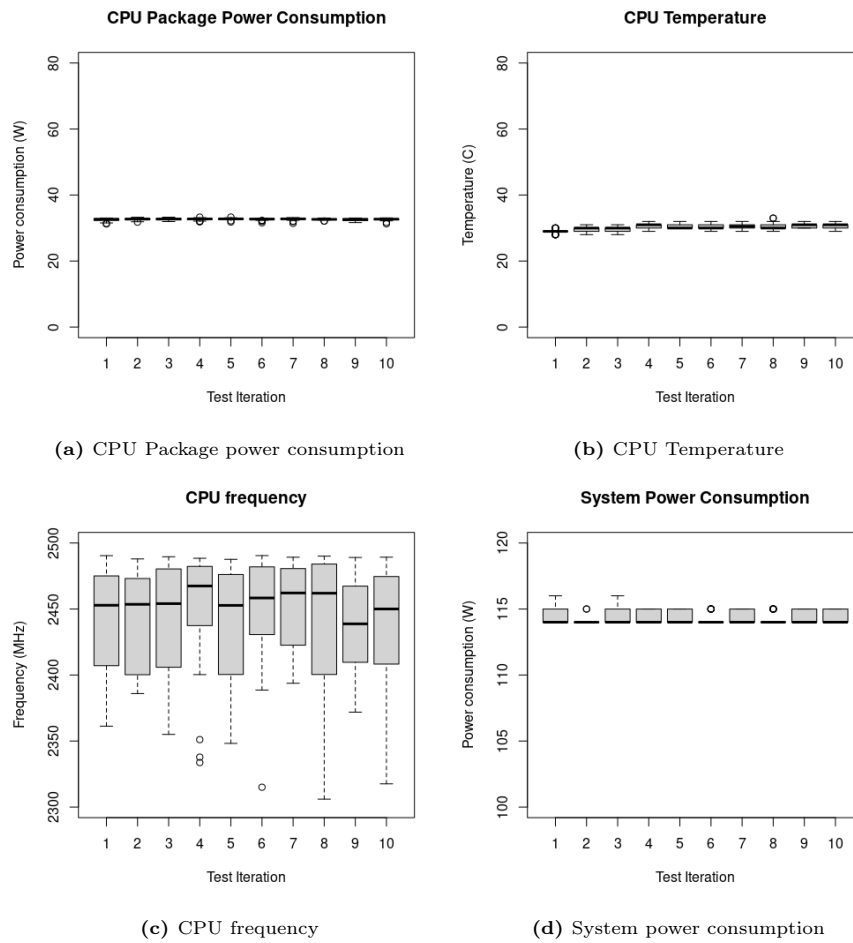


**Figure 29:** Test results running load generator on Docker on 4 CPU cores.

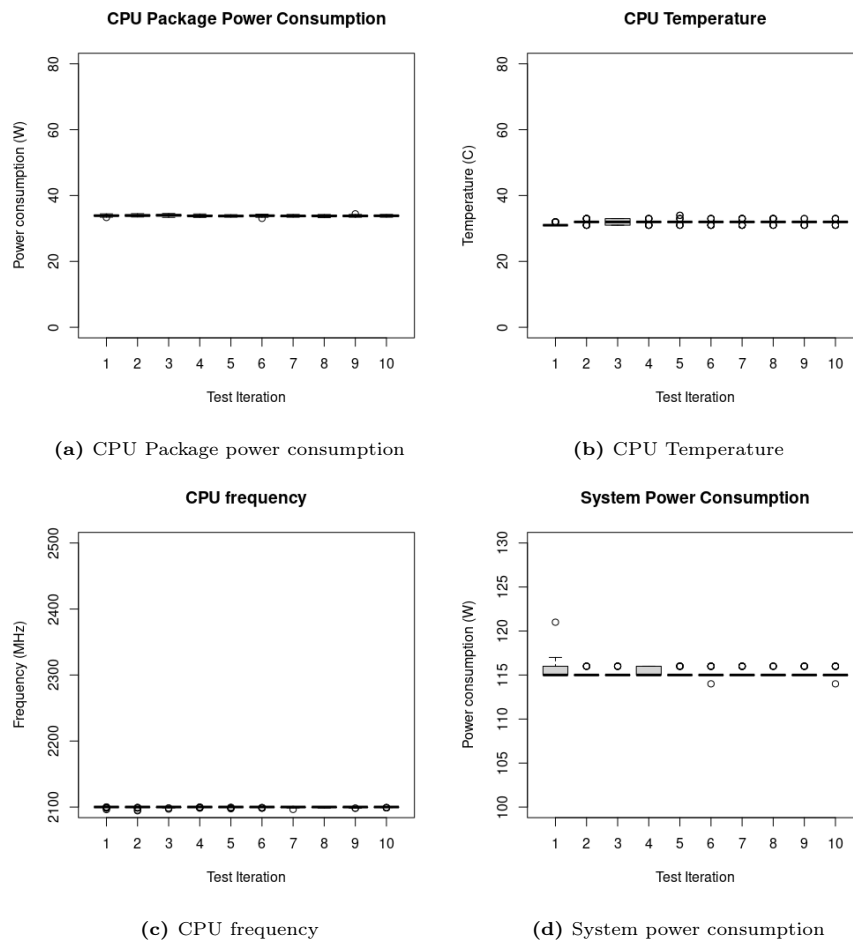


**Figure 30:** Test results running load generator on Docker on 6 CPU cores.

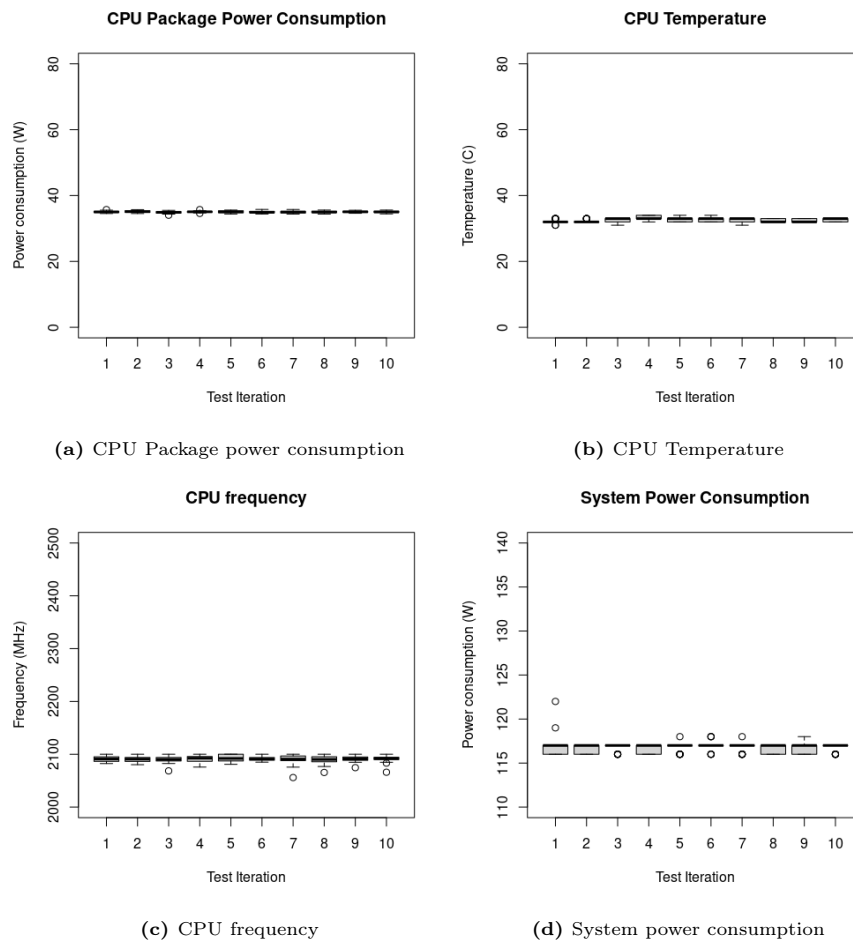
## A.3.3 CPU load with I/O



**Figure 31:** Test results running load generator on Docker on 2 CPU cores.



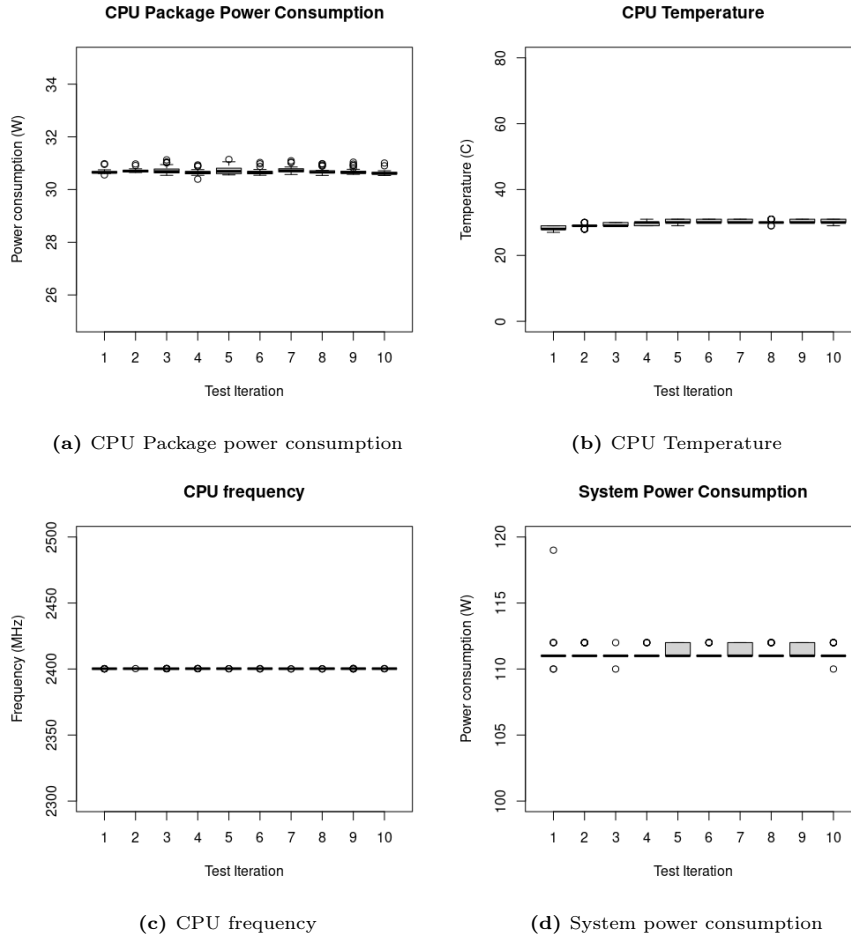
**Figure 32:** Test results running load generator on Docker on 4 CPU cores.



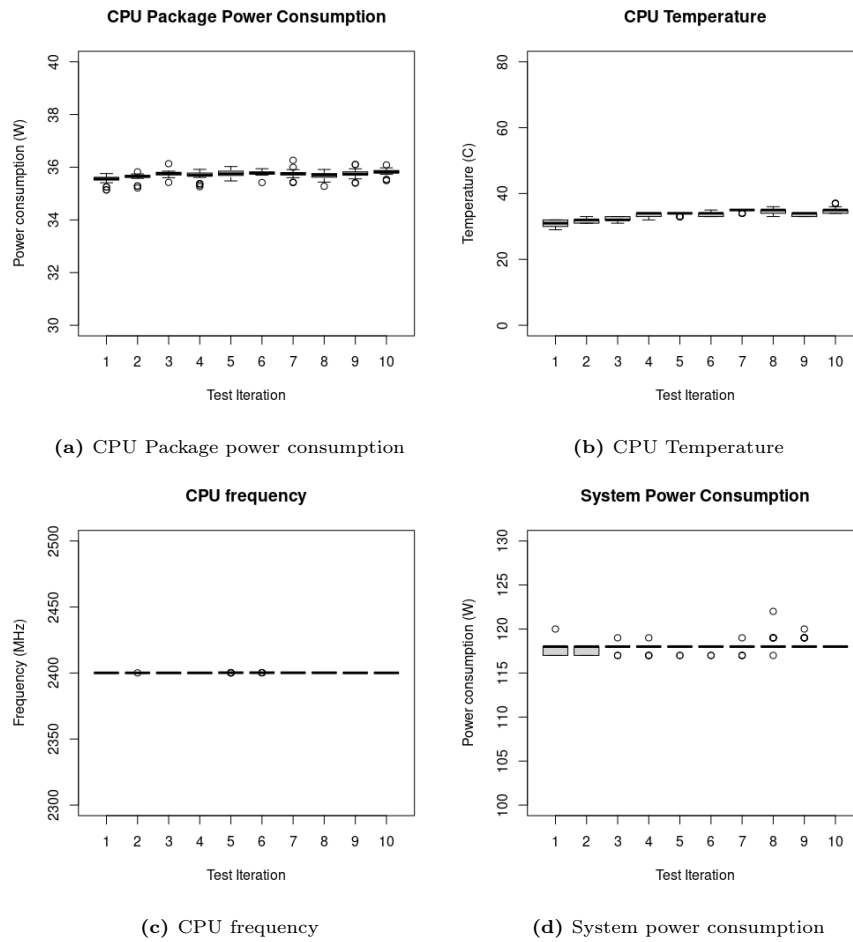
**Figure 33:** Test results running load generator on Docker on 6 CPU cores.

## A.4 Virtual Machine

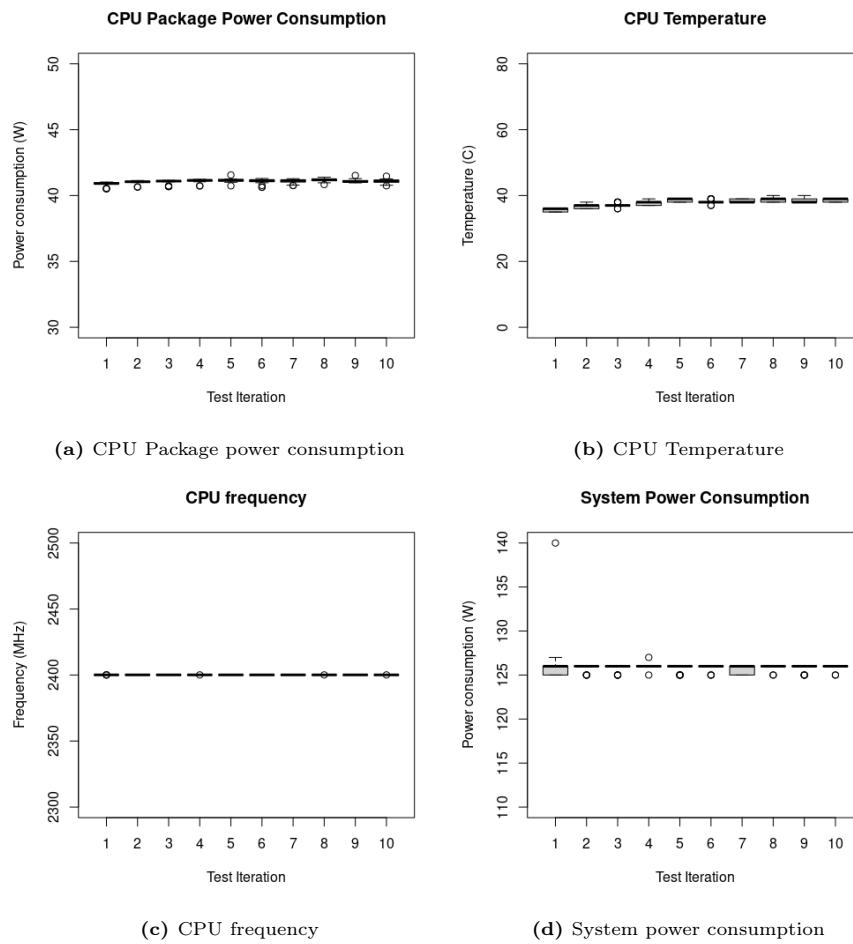
### A.4.1 CPU load without system calls



**Figure 34:** Test results running load generator in a Virtual Machine on 2 CPU cores.

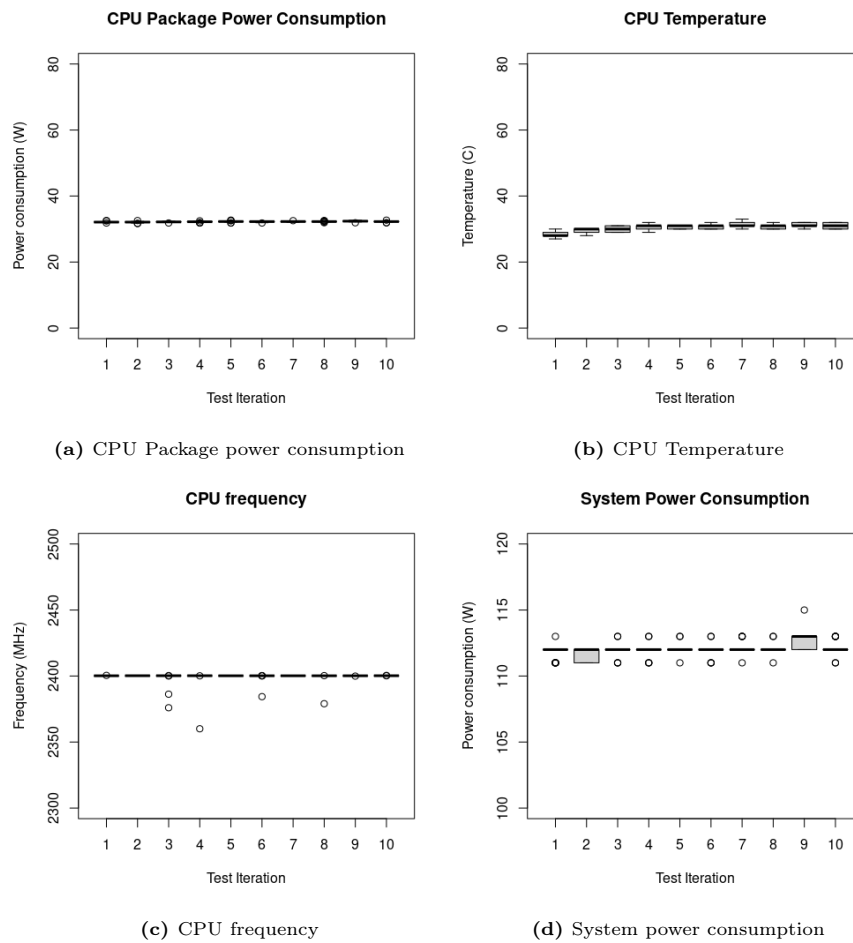


**Figure 35:** Test results running load generator in a Virtual Machine on 4 CPU cores.

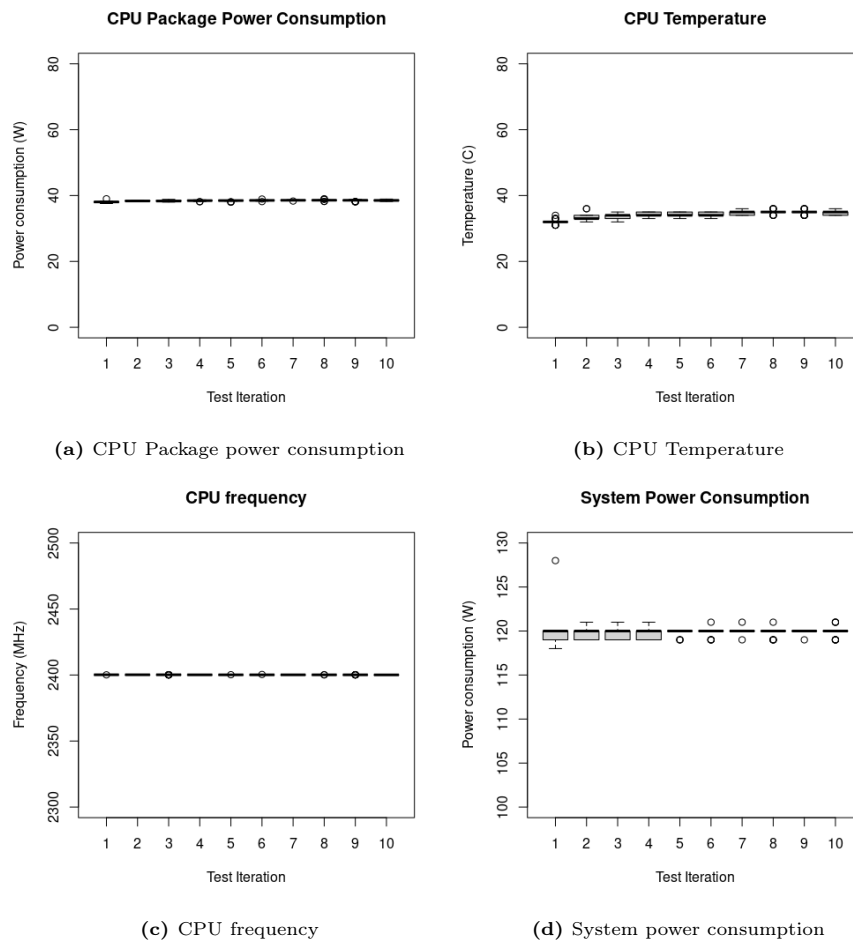


**Figure 36:** Test results running load generator in a Virtual Machine on 6 CPU cores.

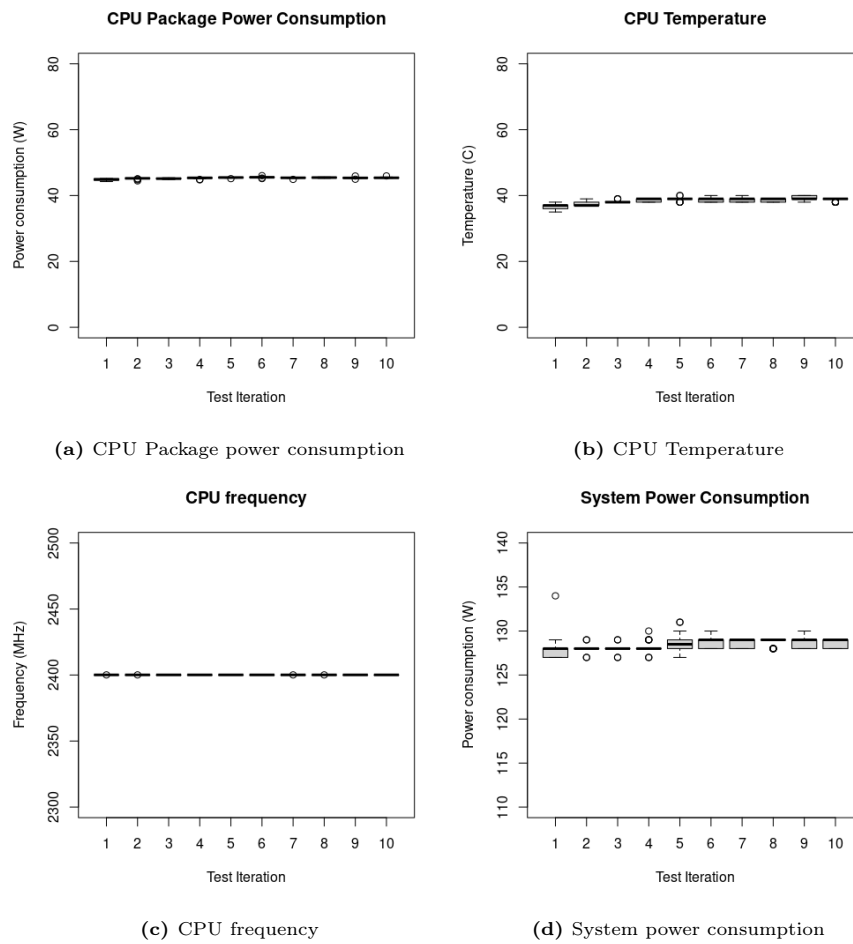
## A.4.2 CPU load with system calls



**Figure 37:** Test results running load generator in a Virtual Machine on 2 CPU cores.

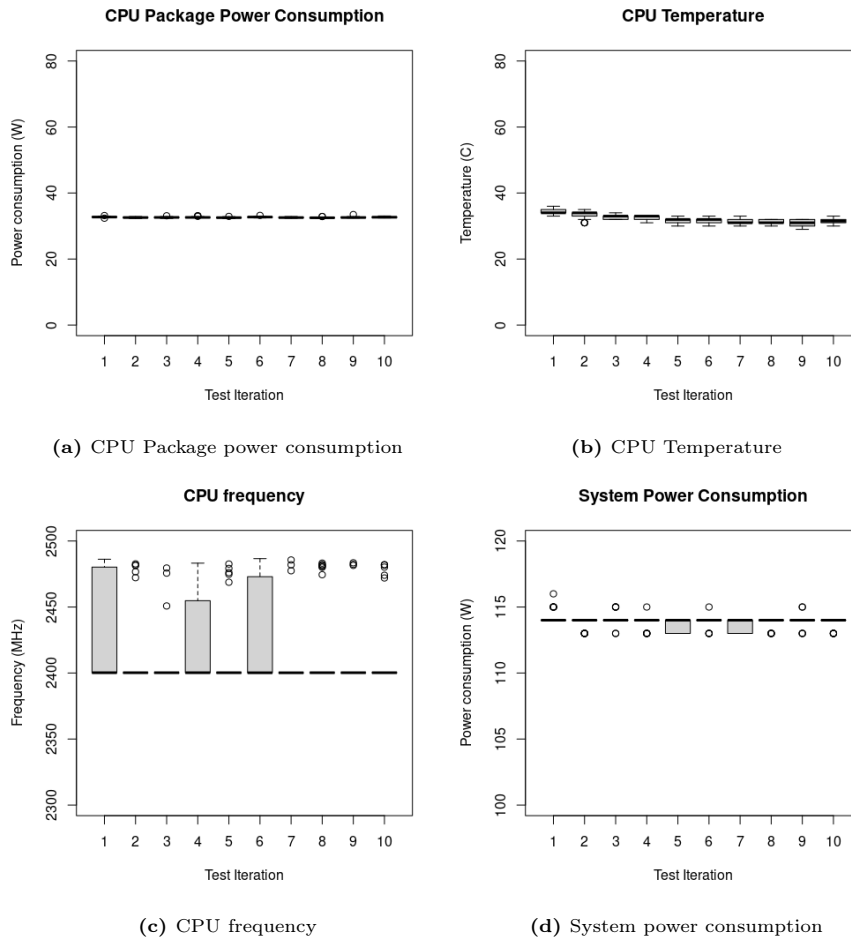


**Figure 38:** Test results running load generator in a Virtual Machine on 4 CPU cores.

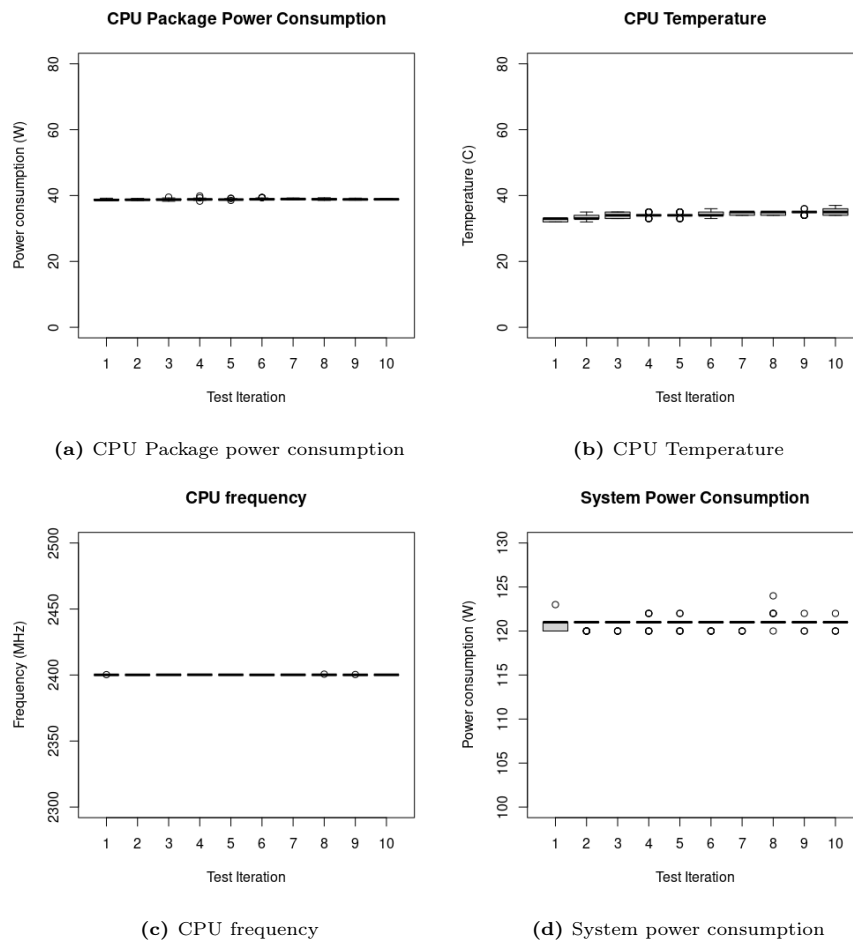


**Figure 39:** Test results running load generator in a Virtual Machine on 6 CPU cores.

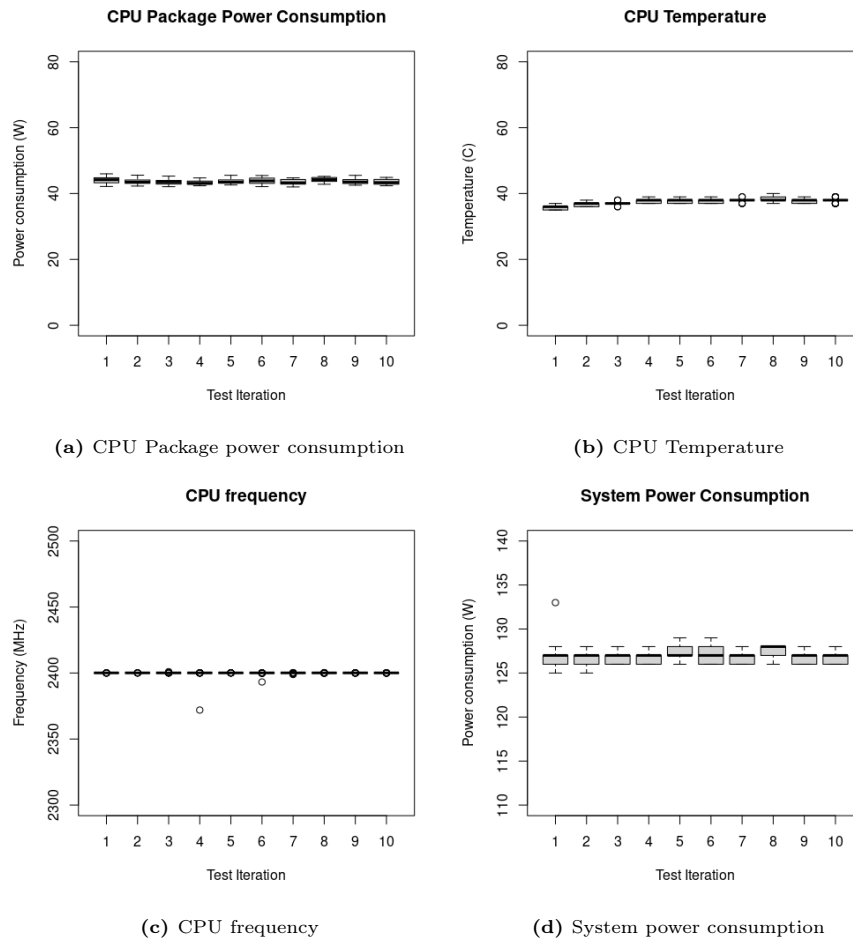
### A.4.3 CPU load with I/O



**Figure 40:** Test results running load generator in a Virtual Machine on 2 CPU cores.



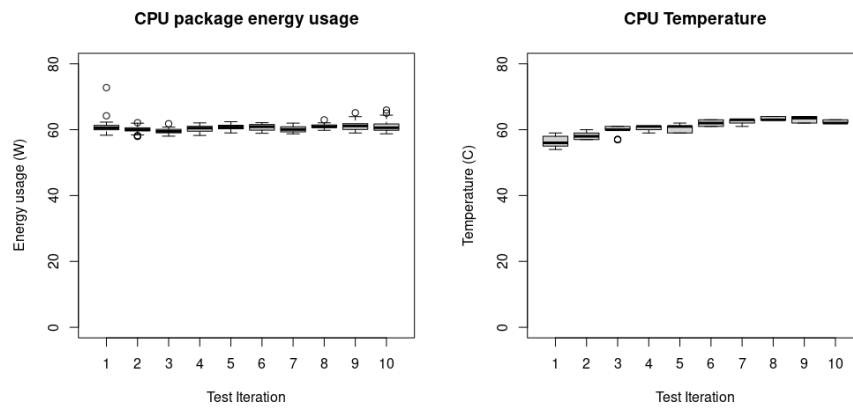
**Figure 41:** Test results running load generator in a Virtual Machine on 4 CPU cores.



**Figure 42:** Test results running load generator in a Virtual Machine on 6 CPU cores.

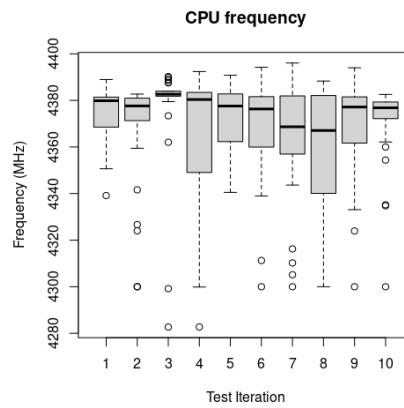
## A.5 Consumer grade CPU

These tests were performed on a desktop PC without the ability to monitor the complete system power consumption.



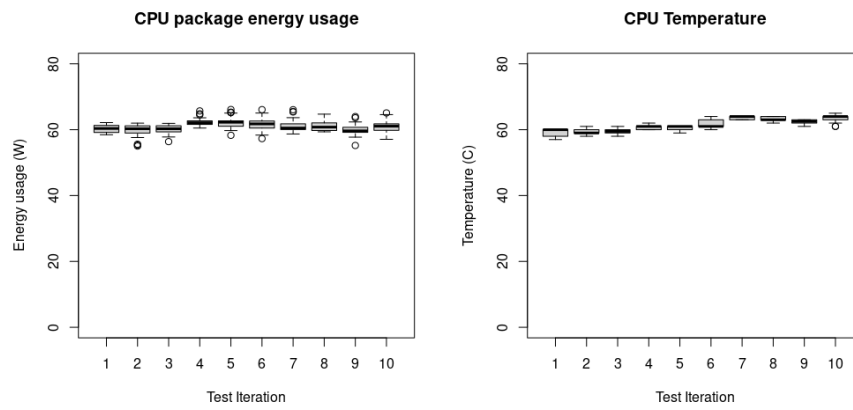
(a) CPU Package power consumption

(b) CPU Temperature



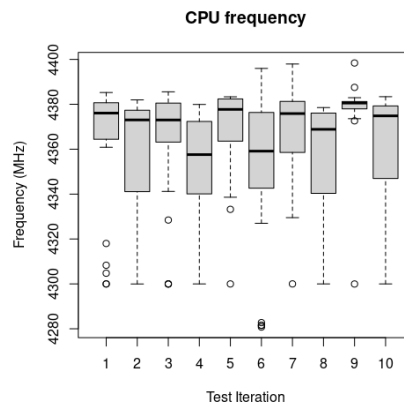
(c) CPU frequency

**Figure 43:** Test results running load generator on bare metal on 4 CPU cores.



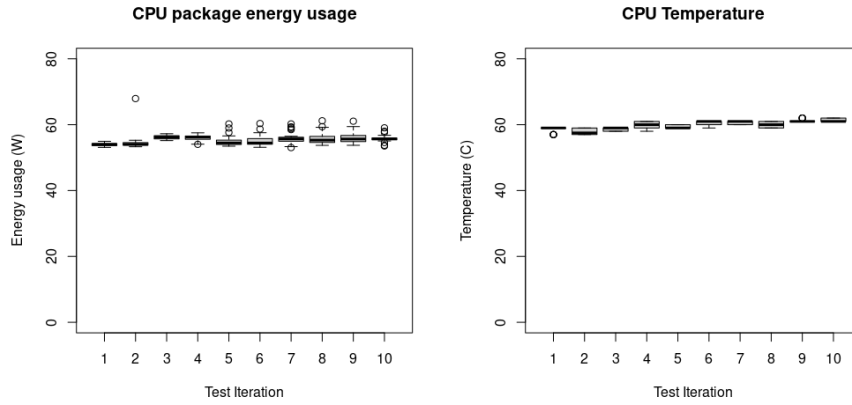
(a) CPU Package power consumption

(b) CPU Temperature



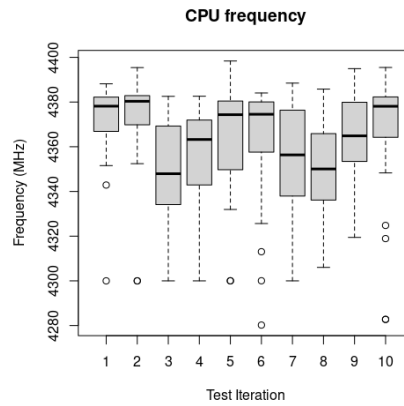
(c) CPU frequency

**Figure 44:** Test results running load generator on Docker on 4 CPU cores



(a) CPU Package power consumption

(b) CPU Temperature



(c) CPU frequency

**Figure 45:** Test results running load generator in a Virtual Machine on 4 CPU cores.

---

## References

- [1] International Energy Agency. *Data Centers and Data Transmission Networks*. URL: <https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks>.
- [2] Mohammad-Mahdi Bazm et al. “Side channels in the cloud: Isolation challenges, attacks, and countermeasures”. In: (2017).
- [3] *cgroups - Linux control groups*. 2023.
- [4] A.P. Chandrakasan, S. Sheng, and R.W. Brodersen. “Low-power CMOS digital design”. In: *IEEE Journal of Solid-State Circuits* 27.4 (1992), pp. 473–484. DOI: [10.1109/4.126534](https://doi.org/10.1109/4.126534).
- [5] The kernel development community. *CPU Idle Time Management*. URL: <https://www.kernel.org/doc/html/v5.0/admin-guide/pm/cpuidle.html>.
- [6] *Containers vs. Virtual Machines (VMs): What’s the Difference?* URL: <https://www.ibm.com/blog/containers-vs-vm/>.
- [7] *Demystifying container vs VM-based security: Security in plaintext*. URL: <https://cloud.google.com/blog/products/gcp/demystifying-container-vs-vm-based-security-security-in-plaintext>.
- [8] *Efficiency*. URL: <https://www.google.com/about/datacenters/efficiency/>.
- [9] Wilbert van Ham. *Computer sleep accuracy*. URL: <https://www.socsci.ru.nl/wilberth/computer/sleepAccuracy.html>.
- [10] Omar Javed and Salman Toor. “Understanding the quality of container security vulnerability detection tools”. In: *arXiv preprint arXiv:2101.03844* (2021).
- [11] Kashif Nizam Khan, Mikael Hirki, and Tapio Niemi. “RAPL in Action: Experiences in Using RAPL for Power Measurements”. In: 2020. URL: <https://api.semanticscholar.org/CorpusID:231834887>.
- [12] Jakub Krzywda et al. “Power-performance tradeoffs in data center servers: DVFS, CPU pinning, horizontal, and vertical scaling”. In: *Future Generation Computer Systems* 81 (2018), pp. 114–128.
- [13] The National Renewable Energy Laboratory. *High-Performance Computing Data Center Power Usage Effectiveness*.
- [14] Daniel Bristot de Oliveira, Daniel Casini, and Tommaso Cucinotta. “Operating System Noise in the Linux Kernel”. In: *IEEE Transactions on Computers* 72.1 (2023), pp. 196–207. DOI: [10.1109/TC.2022.3187351](https://doi.org/10.1109/TC.2022.3187351).
- [15] Arman Shehabi et al. “United states data center energy usage report”. In: (2016).
- [16] *Smart Electricity Meter Accuracy Class*. URL: [https://www.topsmetering.com/blog/smart-electricity-meter-accuracy-class\\_b23](https://www.topsmetering.com/blog/smart-electricity-meter-accuracy-class_b23).
- [17] Paul Townend et al. “Improving data center efficiency through holistic scheduling in kubernetes”. In: *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE. 2019, pp. 156–15610.
- [18] *Types of data centers*. URL: <https://www.ibm.com/topics/data-centers>.

- [19] *What is virtualization*. URL: <https://www.ibm.com/topics/virtualization>.