university of groningen

faculty of science and engineering

Carbon Footprint Monitoring up to Container-Level in Virtualized Environments: A Hardware and Hypervisor-Free Approach

Ties Pol

university of groningen / faculty of science and engineering

University of Groningen

Carbon Footprint Monitoring up to Container-Level in Virtualized Environments: A Hardware and Hypervisor-Free Approach

Master's Thesis

To fulfill the requirements for the degree of
Master of Science in Computing Science
at the University of Groningen under the supervision of
Prof. V. (Vasilios) Andrikopoulos (Computing Science, University of Groningen)
and
dr. B. (Brian) Setz (Lead of Digital Lab, University of Groningen)

Ties Pol (S4597478)

April 11, 2024

Abstract

The popularity of cloud computing and the increasing demand for computing resources have led to a significant increase in energy consumption. In 2015, the electricity consumed by data centers accounted for 0.9% of global energy usage. This is expected to reach 4.5% in 2025. This increase in demand also drives the carbon footprint of computing environments. This thesis starts by conducting a comprehensive literature review, focusing on methodologies for estimating and mitigating the carbon footprint in computing infrastructures, spanning various models and monitoring tools.

Subsequently, this thesis presents an exploratory approach to monitoring carbon emissions down to the level of individual containers. Unlike conventional methods that require hardware, hypervisor, or software access, we try to operate independently of such dependencies. We instead rely exclusively on virtual machine metrics to estimate energy consumption and carbon emissions at the bare metal server level. This result is used to estimate the carbon footprint of the virtual machines and containers in a Kubernetes environment.

The validation of our approach involves assessments to ensure that the calculations align consistently across various levels of abstraction. In addition, we align the energy consumption with the number of requests per second by adding load to a container.

Although the results of inaccessible levels of abstraction approximately align with the load that is added to the system, there is no internal consistency. The results could be useful to get a rough understanding of the carbon footprint. However, additional research is required to explore whether there exists a consistent method for estimating energy consumption in the absence of hardware or hypervisor-level metrics, and thereafter assess the accuracy of the results.

Contents

1 Introduction

Our world has become increasingly reliant on cloud computing, a trend expected to continue growing in the foreseeable future. Over the next two decades, computational capacity is predicted to multiply a million-fold [1]. This exponential growth has brought about a new era of enhanced online connectivity and innovative thinking, with data centers serving as the backbone of an increasingly digitalized society [2].

Cloud computing, essentially offering computing services over the internet, provides a model allowing on-demand access to a shared pool of configurable computing resources [3]. This includes networks, servers, storage, applications, and services, which can be rapidly provisioned and released with minimal management effort, enabling both companies and individuals to utilize high-quality IT infrastructures without large investments in hardware and software. As cloud computing evolves, it incorporates advanced features like virtualization, automation, and data management capabilities [4].

According to the latest forecast from Gartner Inc., worldwide end-user spending on public cloud services grew by 20.7% to $591.8 billion in 2023, up from $490.3 billion in 2022, surpassing the 18.8% growth forecast for 2022 [4]. This highlights that cloud computing is a rapidly growing industry and is expected to continue to grow in the future.

As cloud computing continues to increase its position as a fundamental component of modern operations, the demand for energy to sustain its functionality rises substantially. This is a significant challenge, as the electricity consumed by data centers amounted to 0.9% of global energy consumption in 2015, a figure projected to reach 4.5% by 2025 [5].

This surge in energy consumption results in higher carbon emissions. While the relationship between energy use and emissions is not purely proportional [6], increased energy consumption directly leads to more carbon emissions. Consequently, this contributes directly to global warming, resulting in extreme weather events like typhoons and high temperatures, which adversely affect people's health and quality of life. Therefore, finding effective ways to decrease greenhouse gas emissions during economic development has become a global priority [7]. Given the ongoing energy and climate challenges, it is crucial to focus on ways to reduce and monitor energy consumption and carbon emissions in computing infrastructures [1].

1.1 Carbon Definitions

Before proceeding further, it is essential to understand the terminology used in this thesis regarding carbon emissions and footprint. Section 2.3 will explore more about these terms.

Greenhouse Gas (GHG) Protocol In the domain of carbon emission reporting, the Greenhouse Gasses (GHG) Protocol is used globally and recognized as a standard framework. The mission of the GHG Protocol is to develop internationally accepted greenhouse gas (GHG) accounting and reporting standards and tools and to promote their adoption to achieve a low emissions economy worldwide [8]. It is used to measure and manage greenhouse gas (GHG) emissions from private and public sector operations, value chains, and mitigation actions. This framework classifies emissions into three scopes, each of them representing different sources of carbon emissions [9].

CO_2 Equivalent Carbon dioxide equivalent[1], abbreviated as CO2-eq is a metric measure used to compare the emissions from various greenhouse gases based on their global-warming potential (GWP), by converting amounts of other gases to the equivalent amount of carbon dioxide with the same global warming potential. This ensures that the impact of different gases is expressed in a common unit.

Carbon Intensity The carbon intensity of electricity from the grid worldwide shows how much greenhouse gas enters the air when fossil fuels are burned to make electricity. It is figured out by adding up the carbon intensity of different energy sources, with each one's contribution weighted based on how much of it gets used (like what kind of fuel it is) [10]. Carbon intensity is often expressed in grams of CO2 per kilowatt-hour

[1] https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Glossary:Carbon_dioxide_equivalent

(gCO2/kWh). This means that for every kilowatt-hour of electricity used, a certain amount of CO2 is emitted. Therefore, we can calculate the carbon emissions by multiplying the energy usage by the carbon intensity [9].

Carbon Footprint This article by The Carbon Trust [11] defines carbon footprint as *"A carbon footprint is the total greenhouse gas (GHG) emissions caused directly and indirectly by an individual, organization, event or product"*. It is calculated by summing the emissions resulting from every stage of a product or service's lifetime (material production, manufacturing, use, and end-of-life). Throughout a product's lifetime or lifecycle, different GHGs may be emitted, such as carbon dioxide (CO2), methane (CH4), and nitrous oxide (N2O), each with a greater or lesser ability to trap heat in the atmosphere. These differences are accounted for by the global warming potential (GWP) of each gas, resulting in a carbon footprint in units of mass of carbon dioxide equivalents (CO2e).

1.2 Virtualization

Virtualization will be a key concept in this thesis, therefore important to introduce it here. Virtualization is a technique integral to enhancing system infrastructure efficiency by dividing a system into simpler, more independent modules. Its widespread adoption has made it a common feature in everyday computing. By focusing on consolidation to reduce costs, virtualization seeks to enhance system efficiency and performance [4]. The case study in this thesis involves virtualization at multiple levels of abstraction, including servers, virtual machines, clusters, and containers. Therefore, the following paragraph provides a brief overview of these abstractions.

Servers are often virtualized, which means that the physical bare metal **(BM)** machine, also called a server, is abstracted and shared among multiple virtual machines **(VM)**, also called nodes. The layer between the physical machine and the virtual machines is called the hypervisor. The hypervisor is responsible for managing the virtual machines and their resources, such as CPU, memory, and storage. Often multiple of these virtual machines are then grouped in a cluster **(CLU)**. A cluster abstracts a group of interconnected physical machines, enabling them to function collectively as a unified entity. It facilitates the distribution of workloads across multiple virtual machines for improved efficiency and fault tolerance. Containers **(CON)** abstract the operating system and its dependencies, encapsulating applications in isolated environments. They enable lightweight and portable deployment of software, sharing the underlying host system's kernel for efficient resource utilization. A cluster determines on which virtual machine a container runs. The software **(SOFT)** is the highest level of abstraction and is the application that runs in the container. These abstractions and their abbreviations are used frequently throughout this thesis.

1.3 Case Summary

Efforts to optimize energy consumption and reduce carbon emissions in computing infrastructures are imperative. This thesis delves into the research and development of solutions aimed at creating insights into energy consumption and carbon emissions, particularly in the context of cloud computing. By examining the carbon emissions resulting from server utilization this thesis seeks to raise awareness, facilitate informed decision-making, and drive proactive steps toward mitigating environmental impact.

This thesis focuses on a case study originating from the Digital Lab at the University of Groningen, responsible for applications used in the Computing Science department. The applications, hosted at a private data center managed by the university, require computational resources to operate efficiently. Specifically, this study examines the Themis application, which automates code submission checks by running them in individual containers. Given the energy consumption associated with running these containers, understanding and monitoring carbon emissions becomes imperative.

With both the University of Groningen and the Digital Lab committed to sustainability, the objective is to monitor and visualize carbon emissions resulting from server utilization, especially at the container level within Themis. By comprehensively understanding the extent of carbon emissions released into the atmosphere, stakeholders can make informed decisions and take proactive measures to mitigate environmental impact.

1.4 Problem Definition

The wider configuration space and more heterogeneity of current data center and cloud architectures make software operation more complex [5]. This complexity of data center environments and the rise of cloud computing have made it difficult to monitor and manage energy consumption.

The complexity of our case study is further increased by requirements and constraints coming from Digital Lab. The case study requires the solution to be compatible with OpenStack[2] and Kubernetes. The system needs to support multi-tenancy to enable simultaneous monitoring of containers. Multi-tenancy is understood in various ways depending on the service models [12]. In our case, we adopt the Software as a Service (SaaS) model, where multi-tenancy means that two or more customers utilize the same service or application provided by the Cloud Service Provider (CSP) regardless of the underlying resources [13]. On top of that, a strict requirement is that the solution must monitor at least the container-level abstraction so that the carbon emissions of individual containers can be calculated. In addition, the solution must be able to operate without access to physical hardware, hypervisor, or administrative rights on OpenStack.

Therefore the following research questions arise:

1. *How can the carbon footprint of individual containers be calculated in virtualized environments?*
   This is a broad question, yet it is the main objective that this thesis aims to solve. Since virtualization is a key technology in cloud computing in combination with containers, it is important to see how the related carbon footprint can be calculated in such an environment.

2. *How can the reliability and precision of carbon emission calculations be assessed and validated to ensure consistency and accuracy in results?*
   Ensuring the reliability and precision of carbon emission calculations is crucial for making informed decisions. This question investigates methodologies to validate and assess the accuracy of these calculations.

3. *How to monitor and visually represent carbon emissions at various levels of abstraction and raise users' awareness?*
   This question delves into techniques for real-time monitoring and visual representation of carbon emissions, aiming to raise awareness among stakeholders and facilitate informed decision-making.

1.5 Thesis Structure

The remainder of the thesis is structured as follows: Section 2 provides a comprehensive literature review, focusing on state-of-the-art methodologies for calculating, monitoring, and mitigating carbon emissions in computing infrastructures. Section 3 presents the case study in detail, outlining the context, objectives, and problems. Section 4 discusses the design and implementation of the solution, followed by its validation in Section 5. Finally, Section 6 concludes the thesis by discussing the results, answering the research questions, reflecting on lessons learned, and highlighting avenues for future research.

[2] https://www.openstack.org/software/

2 Background & Related Work

This section investigates the state-of-the-art literature on the relationship between carbon footprint and cloud computing. A large part of this study is dedicated to tools that can monitor the cloud environment. The tools are categorized into system metrics, energy consumption, and carbon footprint monitoring. For each tool, the level of abstraction and the resources it can monitor are gathered. The tools can be found in Section 2.4. Besides the tools, this section also goes into depth about the literature on energy consumption, carbon emissions, and reduction.

2.1 Server Power Modeling

Power Models can be seen as the foundation of software-centric power monitoring and estimation. A power model is a mathematical model that estimates the power consumption of a system based on its resource utilization.

Besides using them to calculate the current power, they are useful for several reasons. First, they can be used to design data center systems. When a data center is designed, it is important to know how much power the complete system will consume. Second, it can help with forecasting trends in power efficiency, some of these models can be used to predict future power consumption. Finally, these can be used to optimize power consumption. However, modeling the exact power consumption behavior of a data center, either at the whole system level or the individual component level, is not straightforward. In particular, data center power consumption patterns depend on multiple factors such as hardware specifications, workload, cooling requirements, types of applications, etc. [14].

Since we are now only discussing the modeling of power consumption and not energy consumption, we will gain a rapid understanding of the conversion from power to energy. Energy (E) represents the cumulative work executed by a system during a specific duration (T), whereas power (P) signifies the speed at which the system performs this work. See Equation 1.

$$E = PT \tag{1}$$

So, we understand that we can find energy by multiplying power by time. However, the complicated part is to find out the power when we only have metrics from the system. This is where power models come into play. Power models help us estimate how much power a system is using based on how its resources are being used. Several of these resources like CPU, memory, and disk are the main parts that use up most of the system's power [15].

For the top level of these power models, they fall into two main categories. One category relies on utilization data provided by the operating system, while the other is based on metrics provided by the hardware. The former estimates power consumption by analyzing the energy data of a specific computer against various utilization metrics. The latter utilizes hardware performance counters.

Server power consumption can be broken down into two main types: static power and dynamic power. Static power remains constant, encompassing leakage currents and idling processes, while dynamic power varies based on active operations and workloads. This division helps in understanding the different factors contributing to overall power usage in servers.

On the second level, these models can be further divided into additive power models and baseline + active (BA) models. Additive power models represent the total power consumption of a server as the sum of its components, whereas BA models assume that the server's idle state already consumes a significant amount of power [14] [9].

BA models can be subdivided into various types such as linear regression, power function, non-linear, and polynomial models. They are also applicable to distributed servers, where each slave employs a BA model. In contrast, additive models require the calculation of each component individually. E.g. the Distributed Energy Monitoring (DEM) tool, which will be discussed later in Section 2.4, facilitates the use of BA models.

In the following enumeration, we will go over the different types of power models in the second level.

1. **Additive Power Models**
   A foundational power model can be found in Equation 2. This model only considers the CPU and memory. It gives a good understanding of what a power model looks like. The equation provides the total energy consumption $E(A)$ by taking the sum of the energy consumption of the CPU $E_{cpu}(A)$ and the memory $E_{memory}(A)$. This power model can be expanded by adding more energy values of other hardware components to the equation. Components like the hard disk ($E_{disk}(A)$), network interface card ($E_{NIC}(A)$), and input/output devices ($E_{I/O}(A)$) can be added to the equation can be added to the equation.

$$E(A) = E_{cpu}(A) + E_{memory}(A) \tag{2}$$

   In Equation 3, we describe the predicted power consumption (Pt) of a server at a specific time t. This prediction takes into account several key factors such as the CPU utilization u_{cpu}, the rate of memory access u_{memory}, the rate of hard disk I/O requests u_{disk}, and the rate of network I/O requests u_{net}. The coefficients of these factors, denoted as C_{cpu}, C_{memory}, C_{disk}, and C_{nic}, are applied to their respective components in the equation. These coefficients are determined through training and are used to predict the power consumption of a server at a specific time.

$$Pt = C_{cpu,n}u_{cpu,t} + C_{memory}u_{memory,t} + C_{disk}u_{disk,t} + C_{nic}u_{nic,t} \tag{3}$$

   Another approach to calculating energy consumption is by dividing the components into two groups called static power and dynamic power. An example is defined in Equation 4. The total power consumption of a server is the sum of the static power P_{fix} and the dynamic power P_{var}. Static power P_{fix} consumption in a server refers to the power used regardless of the server's activity, including energy wasted due to leaks in semiconductor components and the power needed to maintain basic processes, which should be minimized to reduce energy waste but requires improvements at the chip level [14] [9]. Dynamic power P_{var} consumption in a server is determined by activities such as circuit operation, disk drive access (I/O), and workload execution, with a significant portion of power usage allocated to disk, network, I/O, peripherals, regulators, and other circuitry [14] [9].

$$P_{total} = P_{fix} + P_{var} \tag{4}$$

   If we integrate the previously discussed equations with some also presented in this paper [14], then the entire server power model can be seen in Equation 5. Here, n represents the number of VMs running in the node, encompassing various hardware components. $baseline$ denotes the baseline power, empirically determined. The paper from which the above models are obtained showcases over 200 models. While each serves a different purpose, detailing all of them is beyond the scope of this study.

$$P_{server} = a\sum_{k=1}^{n} U_{cpu}(k) + b\sum_{k=1}^{n} U_{mem}(k) + y\sum_{k=1}^{n} U_{io}(k) + ne + baseline \tag{5}$$

2. **Baseline + Active Power (BA) Models**
   Bohra and Chaudhary [16] introduced the *Vmeter* model, designed for estimating the power consumption of virtual machines. This model considers various factors, including CPU, cache, DRAM, and disk utilization. These metrics are estimated using a linear BA approach, where the calculation had to be done manually. However, in a later stadium, this became automatic.

Lin et al. [15] reviewed and compared several popular power models of cloud server components including CPU, vCPU, memory, and hard disk. Interestingly, they consider vCPU because most of the time vCPU is used in data centers. Most of them need training parameters as input. They built a dataset by collecting power data from `SPECpower_ssj2008`[3]. The SPEC Power benchmark is the first industry-standard benchmark that

[3] `https://www.spec.org/power_ssj2008/`

evaluates the power and performance characteristics of single-server and multi-node servers. It is used to compare power and performance among different servers and serves as a toolset for use in improving server efficiency. They used this data to fit different forms of CPU power models and calculated their error distributions [15]. Also, they discussed two types of memory power models. One is based on performance counters and one is based on memory usage. They show that the correlation between memory power and LLC misses is much stronger than the one between memory power and memory usage.

2.2 Cloud Energy Consumption

Dayarathna et al. [14] surveyed in 2016 the state-of-the-art techniques used for energy consumption modeling and prediction for data centers and their components. Interestingly is that they focused on hardware-centric and software-centric power models. Regarding software-centric approaches, they looked at three layers of abstraction. They also split software-centric into performance counter-based and machine learning-based. The survey makes distinctions between modeling and prediction. This is important to keep in mind because there is a clear difference between the two. The paper goes further about how direct power measurement is almost not used frequently. In addition, they claim that hardware performance counters are a more commonly employed approach for energy consumption prediction.

Hardware Performance Counters, sometimes called Performance Monitoring Counters, are special registers provided by modern processors to indicate their state of execution. These registers can obtain profile code performance. They capture the amount of event types that occur. It can monitor hundreds of different performance metrics, including cycle counts, instruction counts for fetch/decode/retire, cache misses, instructions executed, cache misses, branch mispredictions, and more [17].

However, the main focus of Dayarathna et al. [14] was discussing all sorts of energy and power models. They examined more than 200 power models in their survey at different layers of the data center systems in a bottom-up fashion. Some of them are more advanced than others [18] [19]. Some considered more data center components(e.g. CPU, Disk, etc) and some included the booting and halting phase [20]. They concluded that there is not much work done at higher levels of energy consumption modeling. In addition, the accuracy, generality, and practicality of the majority of power consumption models remain open.

Dullak et al. [17] researched different energy models for each data center component. The components that they considered were CPU, Fan, GPU, Memory, Storage, Network Devices, Switches, and Power Supply. This work distinguishes itself from other studies by finding the most efficient energy model for each component and making a recommendation for it. A noteworthy discovery is the significant power consumption of CPUs, which leads to a linear relationship between CPU utilization and server energy consumption. They explain that some models are based on different theories. One is performance monitoring units, also called Hardware Performance Counters as explained before. However, to obtain these counter registers a driver has to be installed because they are not provided to the user space by most operating systems. Another modeling technique Dullak et al. [17] discuss is based on a queuing theory, which is measured by task arrival rate and mean task execution. They elaborate on additional theories and explore various components.

Noureddine et al. [21] outlines the state-of-the-art approaches and tools for monitoring and estimating the energy consumption of software. First, it starts with explaining about existing hardware-based tools. However, they make a clear statement that these approaches are challenging to maintain and upgrade, yet they offer the benefit of achieving good accuracy. Westin et al. [22] has done some work for predicting energy consumption from CPU usage from a containerized application. The author used an empirical simulation and the use of linear regression. However, the proof of concept does not guarantee high accuracy.

To calculate energy consumption some multiplication factors need to be considered in the model. One is `Power Usage Effectiveness (PUE)`. The PUE is useful to present the proportion of energy that is used to operate the IT equipment concerning the total power draw of a facility [23]. This means it gives a score of how energy efficient a data center is, with the lowest possible score of 1 meaning all energy consumed goes directly to powering the servers and none is being wasted on cooling[4]. This score is often given by the cloud provider.

[4] `https://www.cloudcarbonfootprint.org/docs/methodology/#power-usage-effectiveness`

Another factor is `Replication`, which defines the number of replications of data as well as any associated compute and memory resources. This is done to achieve adequate durability and availability for data stores and to ensure better redundancy in the case of service outages[5]. This advantage comes with a cost, as it uses more resources. Therefore, the energy consumption needs to be multiplied by the total number of replications.

2.3 Carbon Emission Fundamentals

This section will provide a comprehensive exploration of the fundamentals of carbon emissions, focusing especially on the definitions introduced in Section 1.1. The first definition, Greenhouse Gas (GHG) Protocol, classifies emissions into three scopes. These scopes are enumerated below:

1. **Direct emissions**, originating from the company, include on-site machinery and power generator emissions; data centers, although not typically emitting these, may have on-site power generators for backup.
2. **Indirect emissions**, stemming from electricity consumption by devices like servers and cooling systems, are the primary focus of this research, in line with Sotos' guideline for Scope 2 emissions reporting, which encompasses all energy consumption.
3. **Remaining emissions**, that are not in scope 1 or 2, account for either a significant share or most of the emissions. The official GHG website[6] provides a standard calculation guide to classify these emissions within scope 3.

However, GHG protocol has an issue with reporting. If deployments are outsourced to the cloud their respective scope 1 and 2 move to scope 3. This means that it will be more difficult to get the total emissions since scope 3 is not required to report. Therefore, some extensions are being made to the GHG protocol called the Clean Energy Emission Reduction (CLEER) protocol and the Science Based Targets Initiative (SBTi). The goal of these extensions is to reduce emissions of their businesses [9]. An additional proposal has been introduced that discusses the subdivision of scope 3 into scope 3 and 4 [24]. In this framework, Scope 3 encompasses the entirety of the supply chain, while Scope 4 focuses specifically on delivery, usage, and end-of-life considerations. Despite its potential benefits, widespread adoption of this extension has not yet occurred.

The term *Carbon Footprint* will be used extensively throughout this thesis. There are multiple versions of the definition, without a coherent accepted definition. E.g. Wiedmann and Minx proposed an official definition of carbon footprint [25]: *"The carbon footprint is a measure of the exclusive total amount of carbon dioxide emissions that is directly and indirectly caused by an activity or is accumulated over the life stages of a product"*. This definition does not consider different greenhouse gases in combination with the GHG protocol. Therefore, this thesis uses the definition outlined in Section 1.1 from [11].

ENTSO-E, the European Network of Transmission System Operators for Electricity, is the association for the cooperation of the European transmission system operators (TSOs). The 40 member TSOs representing 36 countries are responsible for the secure and coordinated operation of Europe's electricity system, the largest interconnected electrical grid in the world[7]. ENTSO-E is responsible for coordinating and ensuring the reliable operation of the electrical transmission system across Europe. It plays a crucial role in facilitating the exchange of electricity across borders and promoting the integration of renewable energy sources into the European electricity grid.

Multiple sources are available for obtaining the current carbon intensity, with many relying on ENTSO-E data. One popular option is Nowtricity[8], which offers real-time emissions data from energy production across countries. This platform presents interactive graphs illustrating current emissions through its website dashboard. Additionally, Nowtricity offers an API for accessing this data.

Another valuable resource for carbon intensity information is Electricity Maps[9], which provides a con-

[5] `https://www.cloudcarbonfootprint.org/docs/methodology/#replication-factors`
[6] `https://ghgprotocol.org/scope-3-calculation-guidance-2`
[7] `https://www.entsoe.eu/about/inside-entsoe/objectives/`
[8] `https://www.nowtricity.com/country/netherlands/`
[9] `https://www.electricitymaps.com`

sistent API for accessing data at hourly intervals for over 160 regions. Users can access this data freely for non-commercial purposes. Previously, CO2-signal[10] served as another source, but it has been integrated into Nowtricity.

The above shows that the relationship between energy consumption and CO_2 emissions is not simply one-to-one. CO_2 emissions are influenced by the energy sources used for generation at any given time. These sources can range from local ones like solar panels to those determined by the regional energy mix. Each source emits a specific amount of CO_2-eq per kWh produced. Therefore, the average CO_2 emissions can fluctuate significantly as different sources are utilized throughout the day [14]. Greenhouse gas emissions from electricity production show considerable variation depending on the time and place. This broad fluctuation in carbon intensity (the average greenhouse gas emissions per unit of energy consumption) or marginal emissions (the extra greenhouse gas emissions per additional unit of electricity consumption) suggests that both the timing and location of electricity consumption significantly impact its overall global warming effect [26].

The state-of-the-art literature on carbon emissions provides various models to calculate carbon emissions. While Haghshenas et al. [6] primarily discussed reducing carbon emissions, they also introduced equations for estimating power and CO_2 emissions. These formulas rely on input parameters, such as the average power consumption of CPUs and GPUs based on their hardware specifications, along with their corresponding average utilization rates.

2.4 Monitoring Cloud Environments

This section delves into the cloud monitoring tools already discussed in the literature and accessible on the Web. Section 2.4.1 will outline tools suitable for system metrics, while Section 2.4.2 will cover tools applicable to energy consumption. Lastly, Section 2.4.3 will dive into tools specifically designed for monitoring carbon footprint.

2.4.1 System Metrics Tools

The tools that are collected in this section focus on performance characteristics, execution metrics, and resource usage. Moreover, these tools do not provide any information about energy consumption. They all have different levels of abstraction and their unique features. A comparison table of the tools that are collected for this purpose can be found in Table 1.

1. **cAdvisor**
   cAdvisor[11] is specifically developed for container orchestrations. The tool can monitor and profile metrics from containers. cAdvisor runs as a daemon inside a concurrent container which could interfere with profiling results. So, it can collect execution metrics system-wide and for individual containers. Since the tool is designed for Docker containers, it can be integrated into a Kubernetes environment.

2. **OProfile**
   OProfile[12] is an open-source profiling framework capable of tracing application execution and evaluating run-time performance. It supports profiling single or multiple processes but has a maximum sampling granularity of 100 ms. When working with software containers, OProfile faces compatibility challenges, requiring additional setup and scripting for monitoring, which can affect profiling accuracy [5].

3. **Ceilometer**
   Ceilometer[13] serves as a data collection service, offering the capability to standardize and convert data from all existing core components within the OpenStack ecosystem. Efforts are also being made to extend support to upcoming OpenStack components. Ceilometer is an integral part of the Telemetry project, and

[10] https://www.co2signal.com
[11] https://github.com/google/cadvisor
[12] https://oprofile.sourceforge.io/news/
[13] https://docs.openstack.org/ceilometer/latest/index.html

its data plays a crucial role in enabling functions such as customer billing, resource monitoring, and alerting across the entire spectrum of OpenStack core components. One crucial disadvantage is that this tool only compatible is with OpenStack.

4. **Monasca**
   Monasca[14] is an open-source multi-tenant, highly scalable, performant, fault-tolerant monitoring-as-a-service solution that integrates with OpenStack. Monasca comes with a monitoring agent for gathering metrics and sending them to the Monasca API. The Agent supports collecting metrics from a variety of sources. First, it can collect system metrics such as CPU and memory utilization. Second, it supports scraping metrics from endpoints provided by Prometheus exporters or Prometheus instrumented applications. Third, the agent can perform checks on OpenStack processes. Just like Ceilometer, Monasca is only compatible with OpenStack.

5. **Venus**
   Venus[15] is an OpenStack project that aims to provide a one-stop solution to log collection, cleaning, indexing, analysis, alarm, visualization, report generation, and other needs, which involves helping operator or maintainer to quickly solve retrieve problems, grasp the operational health of the platform, and improve the level of platform management. Which can include OpenStack logs, operating system logs, cloud platform service logs, and virtualized application-related logs. Again, this tool is only compatible with OpenStack.

6. **OpenTelemetry**
   OpenTelemetry[16] provides a set of tools and interfaces for tracking and managing data regarding software performance and behavior. This data includes metrics, which are measurements taken while your software runs. Each measurement, called a "metric event," not only includes the measurement itself but also records when it happened and any related information. These metrics can also be obtained in a Kubernetes environment since it has an advanced integration with Kubernetes.

   On top of that, OpenTelemetry provides some additional plugins that can be used to monitor the system. One interesting to mention is a collector plugin called *Host Metrics Receiver*[17] that is able to collect metrics about the host system scraped from various sources. This is intended to be used when the collector is deployed as an agent. It can scrape metrics from all sorts of hardware resources. One major advantage is that the plugin is able to collect host metrics from inside a container. Host metrics are collected from the Linux system directories on the filesystem. When it is preferred to collect metrics about the host system and not the container, the host file system needs to be bind-mounted and the path of the root needs to be configured.

7. **Prometheus**
   Prometheus[18] is an open-source system monitoring and alerting toolkit originally built at SoundCloud. After Kubernetes, Prometheus joined the Cloud Native Computing Foundation as well. Kubernetes is also used a lot in combination with Prometheus. The Query language and structure that Prometheus offers are also frequently adopted by many tools that are discussed in this section.

   Prometheus has a Node Exporter[19] that can collect hardware and OS metrics exposed by *NIX kernels, written in Go with pluggable metric collectors. It can collect metrics from the host system. By default, it is not able to collect metrics from inside a container. This can be accomplished by binding the host file system and configuring the path of the root.

[14] https://www.openstack.org/software/releases/zed/components/monasca
[15] https://docs.openstack.org/venus/latest/
[16] https://opentelemetry.io
[17] https://github.com/open-telemetry/opentelemetry-collector-contrib/blob/main/receiver/hostmetricsreceiver/README.md
[18] https://prometheus.io/docs/introduction/overview/
[19] https://prometheus.io/docs/guides/node-exporter/

Name	Abstraction	Depends On	Disadvantages	Resources	Advantage
cAdvisor	BM, VM, CLU, CON				
OProfile	BM, VM		Extra setup needed for containers		
Ceilometer	BM, VM	OpenStack		Mem, CPU, Disk, Network	
Monasca	BM, VM	OpenStack		CPU, Mem	
Venus	BM, VM	OpenStack			
OpenTelemetry HMR	BM, VM, CON				Kubernetes integration
Prometheus NE	BM, VM, CLU, CON				

Table 1: System Metric Monitoring Tools Comparison

2.4.2 Energy Consumption Tools

The following enumeration will discuss various energy and power measuring tools. At the end of this collection, a comparison is presented in Table 2.

1. **PowerScope**
   This application is proposed by [27] to profile the energy usage of applications. It needs a hardware investment to function, but then it can sample the energy usage by process. This means that it does not use estimation, so it is more precise. In addition, it does not offer real-time energy information [21].

2. **pTop**
   This process-level power profiling tool [28] provides power consumption of running processes. It calculates the Joules of four different components, namely CPU, Network Interface, Memory, and Hard Disk. It retrieves resource utilization by accessing the */proc* directory. This tool can be used with only software and can give real-time energy information. One downside of pTop is that it needs hardware interfaces from the vendor of the machine to access power consumption data [21].

3. **PowerAPI**
   PowerAPI[20] can track how much energy applications use as they run, focusing on system processes and software code blocks. It works at various levels of abstraction, like Bare Metal, VM, Container, App, Process, and Code. The estimates come from both hardware and software perspectives, giving insights into CPU and Hard Disk energy usage [21]. For instance, it relies on a Hardware Performance Counter (HWPC) Sensor, which keeps an eye on Intel CPU performance counters and CPU power consumption.

4. **Jalen**
   Jalen[21] is almost similar to PowerAPI. Both tools use power models to estimate the energy consumption of processes and software blocks. However, Jalen measures consumption at a finer level of abstraction. A disadvantage of Jalen is that it is only compatible with Java applications and it is now deprecated. It uses power information provided by the hardware and operating system [21]. The same author of Jalen introduced a new source code monitoring tool called JoularJX. However, this approach is also only compatible with Java applications.

[20] `https://powerapi.org`
[21] `https://www.noureddine.org/research/jalen`

5. **PowerTop**
   PowerTop[22] can diagnose issues with the power consumption of a Linux-based system. This tool also gives an estimation of the energy consumption of software applications and system components [21]. It is possible to monitor processes and show which of them are utilizing the CPU and wake it from its idle states, allowing it to identify applications with particular high power demands.

6. **Energy Checker**
   This paper [21] discussed Energy Checker, an SDK by Intel that gives the possibility to export and import hardware performance counters from an application. These can measure the run-time of a particular event or process and estimate the energy consumption. Nevertheless, it needs a hardware power meter to function, so only applicable in some scenarios. On top of that, the reference link in the paper [21] is not working anymore and the tool is not available on the Intel website.

7. **Joulemeter**
   Joulemeter[23] uses laboratory benchmarks for their model to calculate the energy consumption of hardware resources and software applications. These models learn over time through calibration. Thus, without previous benchmarks, the models cannot be used. It uses multiple power models for measuring the power consumption of servers desktops, laptops, and individual processes [21] [29] [15]. However, this tool is deprecated and no longer available for public download

8. **CloudMonitor**
   CloudMonitor [30] is an energy monitoring tool that does not require additional hardware. It uses a methodology that predicts energy consumption by usage of hardware resources. It uses a linear relationship to estimate the power. Moreover, it can predict precise information about resource usage once a power model has been trained on a particular hardware configuration [29].

9. **DEM**
   DEM was proposed by Lin et al. in 2018 and stands for distributed energy consumption measurement system. They introduce a model for each hardware resource component. Then the system investigates the mathematical relationship between the resource usage of hardware components and the system energy consumption. In other words, DEM uses a Master-Slave architecture. The slave has different components including a hardware detection module that detects the hardware of the local device to obtain the hardware model. The model matching module matches the same or the closest hardware model with energy consumption information according to the underlying hardware. The power estimation module uses the method based on component power models to calculate the energy consumption. Interesting to say is that the results not only show great accuracy on heterogeneous cloud servers but also in measuring the energy consumption of a cluster [29]. This model is categorized under the BA models as discussed in Section 2.1.

10. **RAPL**
    Running Average Power Limit[24] (RAPL) is an interface by Intel to measure energy consumption. The tool is not an analog power meter but uses a software model [22]. Administrative rights and hardware access on the host are needed for this tool to function. In addition, it requires an Intel CPU with Linux or Windows installed. With RAPL it is possible to programmatically get real-time data on the power consumption of the CPU package and its components, as well as of the DRAM memory that the CPU is managing.

[22] https://wiki.archlinux.org/title/powertop

[23] https://www.microsoft.com/en-us/research/project/joulemeter-computational-energy-measurement-and-optimization/overview/

[24] https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html

11. **PAPI**
    In addition to the previous tool, Performance Application Programming Interface[25] (PAPI) can report power and energy measurements through RAPL events. PAPI provides an API for reading hardware performance counters. A major disadvantage is that it needs to modify the code of the application it wants to measure [5].

12. **Containergy**
    The authors Silva-De-Souza et al. of [5] present a new performance evaluation and profiling tool, namely Containergy. This tool can profile container-specific applications while isolating hardware performance metering using control groups.

    Control Groups (cgroups) are a Linux kernel feature that enables the allocation of system resources to processes, managing and monitoring their resource usage. Cgroups provide a mechanism for grouping processes and applying resource constraints, such as CPU, memory, and disk I/O limits. This allows administrators to control and distribute resources efficiently, preventing one process or group of processes from monopolizing system resources. These processes are organized in hierarchical trees [5]. Essentially, cgroups enhance the kernel's ability to regulate and allocate resources, contributing to improved system performance and resource management in virtualized and containerized environments [31].

    Containergy leverages cgroups to organize processes hierarchically, enabling efficient resource allocation. It combines cgroups with namespaces to create lightweight software containers that offer effective process isolation and resource control, all with minimal overhead compared to traditional hypervisor-based virtualization. These containers not only simplify software distribution by packaging software and related files but also facilitate efficient energy and performance data collection through Performance Counters and container logs. The tool ensures precise data matching by utilizing timestamps with microsecond resolution, allowing for accurate analysis of both hardware and software metrics. It does not use code instrumentation, this means however that the tool is not able to extract metrics on specific parts of the code. As a disadvantage, Containergy requires that a container engine Docker, in this case, must be available and operational on the target system. In addition, kernel support for control groups must already be enabled on the system kernel.

13. **Kepler**
    Kepler[26] [27] (Kubernetes-based Efficient Power Level Exporter) is a Prometheus exporter. It uses eBPF to probe CPU performance counters and Linux kernel tracepoints. eBPF[28] is a revolutionary technology with origins in the Linux kernel that can run sandboxed programs in a privileged context such as the operating system kernel. It is used to safely and efficiently extend the capabilities of the kernel without requiring to change kernel source code or load kernel modules.

    These data and statistics can then be fed into ML models to estimate energy consumption by Pods. Currently, Kepler uses power consumption API as RAPL or ACPI. ACPI stands for Advanced Configuration and Power Interface and is an open industry specification establishing industry-standard interfaces for OS-directed configuration and power management on laptops, desktops, and servers [32]. If a device does not support RAPL or ACPI, another way to get power consumption has to be found. Kepler is a Cloud Native Computing Foundation sandbox project.

    Kepler consists of four different modules:

    - **Kepler Model Server** - The main feature of this module is to return a power estimation model. The request that is sent to this server contains data to estimate the power consumption. Data such as the total of a component, input metrics, and accuracy filters.

[25] https://icl.utk.edu/papi/
[26] https://sustainable-computing.io
[27] https://github.com/sustainable-computing-io/kepler
[28] https://ebpf.io/what-is-ebpf/

- **Kepler Estimator** - Client module.
- **Kepler Exporter** - This module exposes metrics about the energy consumption of Kubernetes components such as Pods and Nodes. These statistics are exported in a Prometheus-friendly format. Besides energy consumption, it exports a variety of container metrics as well.
- **Kepler Power Model** - Kepler provides three modeling approaches. The default model is *Power Ratio Modeling* which uses the ratio over the total summation of power. This one can be used when the total power is known. Second, *Power Estimation Modeling* estimates power by using input features of the trained model. This model can be measured on three levels: node total power, node internal component, and pod power. Lastly, *Pre-trained Power Models* has for some deployment scenarios pre-trained models that can be used[29].

14. **Scaphandre**
    Scaphandre[30] serves as a monitoring agent exclusively designed for tracking energy consumption metrics. Its primary function is to reveal the power usage of individual processes on a computer. The tool effectively executes two key tasks: first, it gathers and pre-computes the power consumption metrics, and second, it publishes this data. By employing Prometheus[31] for metric scraping and Grafana for dashboard creation, users can visualize the collected data. Moreover, Scaphandre offers interaction at various levels of abstraction, including the bare metal host, containers themselves, or the container orchestrator.

    Scaphandre aims to solve the problem that virtual machines usually do not have access to power metrics. By enabling communication between a Scaphandre instance on the hypervisor/bare metal machine and another one running on the virtual machine. The Scaphandre agent on the hypervisor will compute the metrics meaningful for that virtual machine and the one on the VM access those metrics to allow its user/administrator to use the data as if they had access to power metrics in the first place (as if they were on a bare metal machine). This enables breaking opacity in a virtualization context with access to the hypervisor, or in a public cloud scenario where the provider uses Scaphandre on its hypervisors[32].

    Although Scaphandre offers various methods for monitoring power consumption, there are certain limitations to consider. It currently enables direct measurement of hardware through software interfaces. Furthermore, as of the time of writing this thesis, the official website indicates plans to introduce a feature for estimating hardware metrics. A crucial aspect discussed in a GitHub thread[33] revolves around the essential requirement of establishing a lookup database containing specifications of hardware components. This database would facilitate the estimation of power consumption by correlating these metrics with resource consumption metrics such as CPU or RAM usage.

2.4.3 Carbon Footprint Tools

This final section will focus on tools that are specifically designed to monitor carbon emissions or carbon footprint. We have only found one tool in the literature for this section, which is Cloud Carbon Footprint (CCF). Hence, there is no comparison table for this section and we discuss the tool here in detail.

CCF[34] is an open-source tool that provides tooling to measure, monitor, and reduce cloud carbon emissions. This data creates visibility of the usage of cloud resources and the associated carbon emissions. This tool uses best practice methodologies to convert cloud utilization into estimated energy usage and carbon emissions, producing metrics and carbon savings estimates. The tool can be used with two types of data input but uses

[29] https://github.com/sustainable-computing-io/kepler-model-db

[30] https://github.com/hubblo-org/scaphandre

[31] https://prometheus.io/docs/introduction/overview/

[32] https://hubblo-org.github.io/scaphandre-documentation/how-to_guides/propagate-metrics-hypervisor-to-vm_qemu-kvm.html

[33] https://github.com/hubblo-org/scaphandre/issues/25

[34] https://www.cloudcarbonfootprint.org

Name	Abstraction Level	Resources	Pros	Cons	Depends On	Last Update
PowerScope	BM					
pTop	BM	CPU, Disk, Network, Memory				
PowerAPI	BM, VM, CLU, CON, SOFT	CPU, Disk				
Jalen	BM, SOFT	CPU, Disk		Only Java		2021/09/20
PowerTop	BM			Only Linux		
Energy Checker	BM			Needs hardware power meter		
JouleMeter	BM					
CloudMonitor	BM, VM	Memory, CPU, Disk, Network				2013/11/29
DEM	BM, VM, CLU	CPU, Memory Hard Disk		Only heterogeneous cloud		
RAPL	BM, VM					
PAPI	BM, VM			Needs additional hardware		
Containergy	CON				Docker	
Kepler	BM, VM, CLU, CON				Kubernetes, Prometheus	
Scaphandre	BM, VM, CLU, CON		Works with Kubernetes			

Table 2: Cloud Energy Consumption Tools Comparison

the same methodology for both. First, it can be used with billing data from cloud providers. The application pulls usage data from major cloud providers and converts it into energy usage and carbon emissions. Second, it can be used with manual input data. CCF called this method On-Premise[35]. With this approach, it is possible to make estimations without billing information. However, until writing this thesis, the tool only considers CPU and memory usage. Nevertheless, it only needs the machine name to do its calculations. CCF can map the machine name to the associated micro-architecture and leverage a specification database to determine the average compute and memory coefficients.

Currently, CCF exclusively supports on-premise estimations for Compute and Memory usage, omitting operational emissions beyond these parameters. However, there are plans to integrate embodied emissions for on-premise scenarios in the future[36]. The on-premise estimation process mirrors the main methodology[37] of

[35] https://www.cloudcarbonfootprint.org/blog
[36] https://www.cloudcarbonfootprint.org/docs/on-premise/
[37] https://www.cloudcarbonfootprint.org/docs/methodology

the tool but relies on manual input instead of Cloud Provider APIs. Despite its alignment with Compute and Memory usage, it lacks support for network and disk usage and cannot assess virtual machines. Moreover, it requires metrics from entire physical machines, posing challenges for users employing VMs in data centers. Nevertheless, the tool showed confidence as a working energy and carbon footprint model, powered by its open-source nature and active development community.

2.5 Reducing Carbon Emissions

Haghshenas et al. [6] investigated how we can reduce emissions by using the iterative training process and the varying intensity of CO2 signals from the power grid. They introduced the idea of migrating computing jobs to geographical locations with the lowest carbon intensity. Building upon the ideas presented in [10] [26], Haghshenas et al. [6] proposed two mechanisms for shifting load to when and where low-carbon energy is available, namely *suspend_resume* and *moving_between_clouds*. Of the two, the latter showed more promising results in emission reduction.

Another strategy for reducing energy consumption involves leveraging stranded power, as outlined in [9]. This initiative, termed Zero Carbon Cloud (ZCCloud), aims to utilize renewable energy that would otherwise go to waste. ZCCloud achieves this by situating data centers near renewable energy sources, thereby eliminating carbon emissions associated with energy production. This paper [33] created a concept called *Follow the Renewable*. This approach aims to establish data centers near renewable energy sources, optimizing job scheduling to prioritize the use of green energy over traditional sources. Additionally, heat generated by data centers can be repurposed by supplementing it with higher-quality heat, which can then be distributed to nearby households.

A Carbon Kubernetes Scheduler has been created and put into action by James et al. [10]. It moves workloads to areas with reduced carbon dioxide emissions. In other words, it migrates energy consumption to countries with the lowest carbon intensity in real-time. This paper by Xu and Buyya [33] also considers shifting workloads among multi-cloud locations in different time zones. The results show that their approaches, which apply workload shifting, can reduce around 40% carbon emissions in comparison to the baseline while ensuring the average response time of user requests. This is a significant reduction in carbon emissions. This method is similar to the *moving_between_clouds* mechanism proposed by Haghshenas et al. [6].

Radovanovic et al. [26] took another approach by delaying workloads temporally, so that workloads can run at moments with a lower carbon intensity and therefore minimize the carbon footprint. This is done by using Google's Carbon-Intelligent Computing System. The paper shows effective results when delaying the execution of temporally flexible workloads to greener times. This technique is identical to the *suspend_resume* mechanism proposed by Haghshenas et al. [6].

The following paper by Kaur et al. [34] introduces a Kubernetes-based energy and interference-driven scheduler. So, it is not delaying workloads but it schedules it beforehand. They designed a composite controller, called KEIDS. This controller can use maximal green energy leading to minimal emission of carbon footprints, optimal performance, minimal interference, and energy minimization.

This article by Radersma [1] provides some recommendations for software engineers about how they can optimize software to reduce their carbon footprint. It includes choosing a green language, comparing energy usage between different versions, and using GPUs over CPU. Although GPU uses more energy it can be more efficient by good parallelisation. Further, it recommends recalculation over retrieving from (local) storage. Finally, it advises choosing the greenest cluster, since it can differ in twofold.

3 Case Study

This section presents the case study of this thesis and is structured as follows. First, the context of the case study is provided in Section 3.1. The objective is given in Section 3.2, followed by the problem definition in Section 3.3. A technology assessment is conducted in Section 3.4, assessing the tools that can be utilized to achieve the objective. After that, different alternatives are discussed in Section 3.5. The setup that is used for testing and implementation is explained in Section 3.6. Finally, the potential tools are tested in Section 3.7.

3.1 Context

Digital Lab is a team within the Computing Science department at the University of Groningen. It is responsible for applications used by the faculty. Among these applications are *Virtual Labs*, facilitating access to computational resources for courses, and *Repo Man*, which simplifies the creation of source code repositories.

A particular relevant application is *Themis*, which automates code submission checks by executing and compiling the submitted code. These submissions, provided by students for specific assignments, undergo evaluation within a container set up by Themis. The application then compares the container's output with the expected result. Themis is used by half of all the Computing Science courses, processing roughly 120.000 submissions annually.

Themis has an extensive infrastructure, illustrated in Figure 1. It leverages OpenTelemetry to monitor the submission pipeline, tracking metrics such as execution time. Data storage is managed through Minio database buckets, organized by course editions, assignments, and groups. Metadata is stored in MariaDB. Additionally, Themis utilizes Kafka for submission queuing. Thus, the container for the code submissions is surrounded by a diverse array of techniques and resources.

These applications from Digital Lab are hosted at Coenraad Bron Center, the University of Groningen's data center. Infrastructure management is facilitated by OpenStack, granting Digital Lab access to resources via the OpenStack API. OpenStack serves as a cloud operating system, controlling computing, storage, and networking resources across the data center. Its dashboard empowers administrators to oversee resource provisioning while enabling users to manage resources via a web interface. In addition to standard infrastructure services, OpenStack offers orchestration, fault management, and service management functionalities to ensure application availability.

One might imagine that conducting all of these checks consumes a significant amount of server computation, which translates into energy consumption. The energy usage of the servers is not only important for the University of Groningen, but also for the environment. The University of Groningen is a public institution and has a responsibility to be sustainable.

The computational requirements of these applications lead to a considerable amount of energy being used by servers, which in turn contributes to carbon emissions. As a public institution, the University of Groningen is committed to sustainability[38], recognizing its responsibility to minimize environmental impact alongside operational efficiency.

3.2 Objective

Given the University of Groningen's and Digital Lab's commitment to sustainability, it is practical to monitor the carbon emissions produced by the servers utilized by various applications. By understanding the extent of carbon emissions released into the atmosphere, proactive steps can be taken to mitigate the environmental impact.

The primary goal of this thesis is to track and visualize carbon emissions arising from the utilization of Themis. Specifically, the objective is to monitor emissions up to the level of container abstraction. As individual containers rely on different components directly and indirectly linked to a code submission, it is also interesting to explore other levels of abstraction to grasp the energy consumption of the infrastructure comprehensively.

[38] https://www.rug.nl/about-ug/profile/facts-and-figures/duurzaamheid/roadmap-sustainability?lang=en

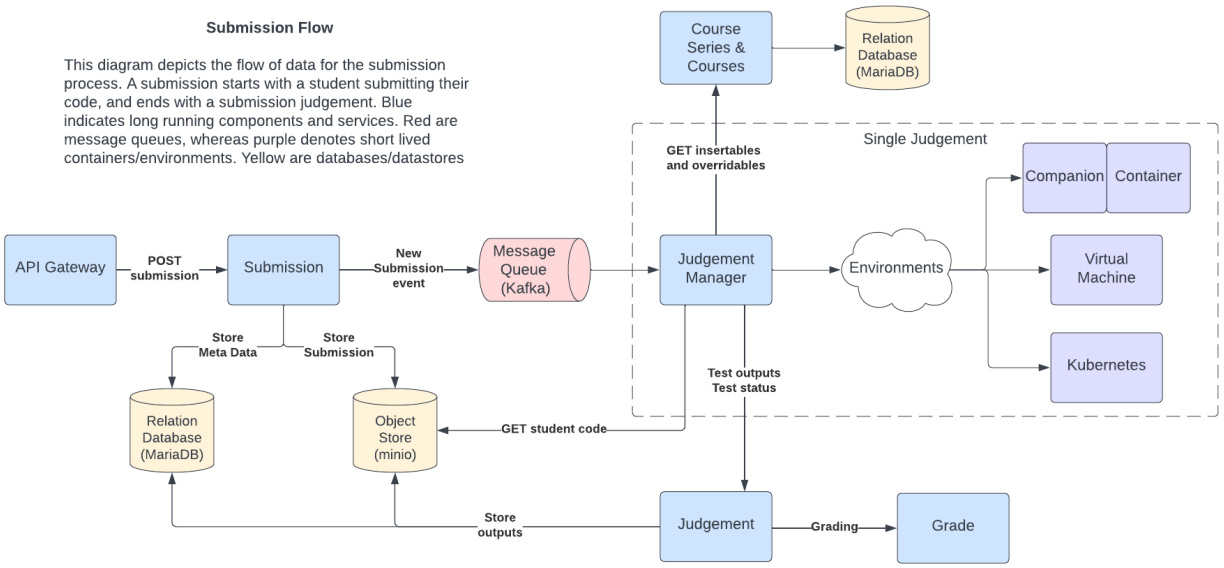


Figure 1: Themis Architecture

A container that runs a submission is only active for a specific timeframe, and it is essential to measure the container's carbon footprint during this period, providing a precise insight into the carbon footprint of a specific code submission.

Furthermore, this project delves into determining the most effective approach to visually represent energy consumption and carbon footprint data. This aspect ensures that the findings are understandable and actionable for making informed decisions. This will give Digital Lab insight into the carbon footprint of their application, Themis.

3.3 Problem Definition

Several challenges must be overcome to achieve the objective. Various requirements and constraints need consideration, originating from Digital Lab's design choices and infrastructure. These requirements are essential for the solution to be compatible with Digital Lab's infrastructure and to monitor the energy consumption of multiple code submissions simultaneously. The constraints, on the other hand, are imposed by the infrastructure used by Digital Lab, over which they have no control.

3.3.1 Requirements

- **Frameworks** - The solution must be compatible with OpenStack and Kubernetes, as Digital Lab utilizes these for infrastructure management.

- **Multi-tenancy** - The solution must handle multi-tenancy to differentiate between multiple code submissions and monitor their energy consumption simultaneously. With 120.000 submissions annually, this capability is crucial.

- **Container Level** - The solution should monitor at least the container level to track the carbon footprint of specific code submissions running in single containers.

3.3.2 Contraints

- **Hardware** - Physical access to the hardware is unavailable as Themis operates in a data center with restricted access. This means that it is not possible to measure the energy consumption of the hardware directly. The hardware is owned by Digital Lab. However, they have six servers in total with three different hardware specifications. So there are three groups of two with the same hardware specifications. To conclude, there are specifications of the hardware available, but there is a chance that we do not know which resources are used at a specific moment.
- **Administrative rights** - Digital Lab has no administrative rights on OpenStack, limiting actions to those possible with ordinary user permissions. While they can configure virtual machines and therefore have access to the kernel of the VM, installing certain plugins or advanced settings is restricted.
- **Hypervisor access** - Digital Lab lacks access to the hypervisor, preventing software installation at that level. This gap between the hardware and virtual machines complicates synchronizing hardware utilization with different levels of abstraction.

The challenge lies in whether we can still measure the energy consumption of code submissions given these constraints and requirements.

3.4 Technology Assessment

This section conducts a technology assessment of tools that can be utilized to achieve the objective with the limitations of the problem considered. The tools gathered in Section 2 will be evaluated to determine which one best suits the needs of Digital Lab.

The assessment will follow a bottom-up approach in comparison to the order of related work. The related work is structured into metrics, energy consumption, and carbon footprint, respectively. This order is chosen because if a potentially applicable carbon footprint technique is identified, we can then search for missing components in the subsequent tools that can serve as input for the carbon footprint technique. Similarly, missing pieces related to energy consumption can be found in the metrics. Each tool has a brief minimum requirement at the end written in italics.

We will evaluate the tools to determine if they can provide information on energy consumption rather than just carbon emissions. Once we have data on energy consumption, we can then proceed to calculate the carbon emissions by multiplying the energy consumption with the carbon intensity of the electricity grid.

In total, two assessment tables are provided besides the enumeration below. The same abbreviations used for the abstraction levels in the related work are applied here, although the software (SOFT) abstraction is excluded from this assessment. First, Table 3 presents the tools alongside checkboxes indicating whether they meet each requirement. Second, Table 4 offers an overview of each tool's compatibility with the constraints grouped by different levels of abstraction. A cross in a checkbox means that the tool can be used despite constraints. A tilde ($\sim$) indicates that the tool can monitor that level when there is no constraint.

1. **Cloud Carbon Footprint (CCF)** - This tool can be used to estimate the carbon footprint and energy consumption based on the hardware specifications. It can only be used to estimate at BM level.

   Requires hardware specifications as a minimum to function. For better estimations, it needs resource utilization data.

2. **Scaphandre** - This is a promising tool since it provides a way to integrate with Kubernetes. However, it requires to have Scaphandre installed on the hypervisor to make it work with virtualized environments. When Scaphandre is installed on the hypervisor, it can expose power readings to the virtual machines and containers that are running on the hypervisor. This enables monitoring power consumption at virtual machine and container levels[39]. Scaphandre uses underwater PowercapRAPL sensors. This means that

[39] https://hubblo-org.github.io/scaphandre-documentation/explanations/how-scaph-computes-per-process-power-consumption.html

this tool can only be used in two cases. One is that if Digital Lab has access to the hypervisor, they can install Scaphandre by themselves. The other case is that the hypervisor already has Scaphandre installed.

Requires access to the hypervisor, so Scaphandre can be installed.

3. **Kepler** - Another promising approach is Kepler, which focuses on Kubernetes, hence its name Kubernetes-based Efficient Power Level Exporter. It is designed to measure energy consumption at the container level. Kepler utilizes the energy consumption measurement capability of RAPL. This tool requires access to the operating system kernel, facilitated by eBPF, to obtain power consumption data. With this access, it leverages hardware performance counters, cgroup, and sysfs to measure all processes/threads on the specified CPU. While Kepler can measure or estimate energy consumption at all considered abstraction levels, it needs a power model to function. Either this model is trained initially on the hypervisor or a pre-trained model is used[40]. This power model is then utilized in the `Kepler Model Server` module as explained in Section 13.

   Requires access to the hypervisor for initial model training or a pre-trained model with the same deployment scenario.

4. **Containergy** - This tool is capable of measuring energy consumption specifically at the container level, rather than at other levels. It focuses on isolating hardware performance counters to measure the energy usage of a container by utilizing control groups. Since access to the host kernel and hardware performance counters is necessary, this tool cannot be employed for measuring energy consumption.

   Requires access to the hypervisor for cgroups, namespaces, and hardware performance counters.

5. **PAPI** - This tool is an addition to RAPL. It is an API that can be used to access the hardware performance counters. This tool alone cannot measure energy consumption, the underlying tool is RAPL. Therefore, we continue with RAPL.

   Requires access to the hypervisor to gain kernel-level access to hardware performance counters.

6. **RAPL** - With this tool, it is possible to get real-time power consumption of the CPU on BM & VM level. It requires administrative rights on the (guest) OS. In addition, the CPU vendor must be Intel. Digital Lab has three AMD CPUs.

   Requires access to the hypervisor to retrieve data from the host CPU and a CPU of the vendor Intel.

7. **DEM** - The DEM can gauge energy usage at the cluster level. For estimating energy consumption, the Hardware Detection Module requires access to the `/proc` virtual directory to gather raw hardware data. In the Linux version, the slave component accesses files such as `/proc/stat`, `/proc/meminfo`, and `/sys/block/sda/stat` to retrieve information on CPU, memory, and disk usage. Consequently, this tool is solely functional when the `/proc-filesystem` is activated in the kernel and has permission to access the host OS kernel.

   Requires access to the hypervisor to gain kernel-level access to the /proc directory.

8. **CloudMonitor** - CloudMonitor provides this capability, by initially collecting "training" information from PDUs to ascertain the connections between resource consumption and power used [30]. This means that it first needs access to the previous data of the host machine. Thus, access to the BM is required.

   Requires access to the hypervisor for previous PDU data of the host machine.

9. **JouleMeter** - This tool, developed by Microsoft, can estimate the energy consumption of various resources used in a computer. It operates at the bare metal abstraction level, meaning it delves into the fundamental aspects of computation. However, the models it employs are derived from calibration, which poses a limitation in terms of flexibility since power models cannot be established without prior

[40] https://www.cncf.io/blog/2023/10/11/exploring-keplers-potentials-unveiling-cloud-application-power-consumption/

laboratory benchmarks. Unfortunately, the public release of Joulemeter has been deprecated and is no longer available for download[41].

Deprecated.

10. **Energy Checker** - Although this tool can measure energy consumption on a bare metal level, it requires a hardware power meter to measure the energy consumption.

    Requires a hardware power meter to be installed.

11. **PowerTop** - This tool is designed to measure energy consumption on a bare metal level. However, to use this tool an additional package is needed on the host machine.

    Requires access to the host machine to install a package.

12. **Jalen** - Jalen is a great tool to measure energy consumption at the source code level of Java applications. Unfortunately, it is only compatible with Java applications. On top of that, it is deprecated.

    Requires the use of Java applications only and is deprecated.

13. **PowerAPI** - Currently, PowerAPI proposes one Sensor: HWPC. In particular, it exploits the perf API of the Linux kernel. It is only available on Linux and needs to have root access to be used. The sensor can not be used in a virtual machine, it must have access (via Linux kernel API) to the real CPU register to read performance counter values[42]. It is possible to install the formula by hand in a Virtual Machine if needed, but then several metrics from the hardware are still needed to be able to calculate the energy consumption. This means that it needs access to the kernel of the host OS to be able to measure energy consumption.

    Needs access to the hypervisor to gain kernel-level access to hardware performance counters.

14. **pTop** - This process-level power measurement tool can measure energy consumption on a variety of different abstractions. However, it requires access to the `/proc` directory to be able to measure energy consumption. Being a service of the operating system, pTop runs at the kernel level and provides energy consumption data from all applications and processes running in the system [28].

    Needs access to the hypervisor to gain kernel-level access to the /proc directory.

15. **PowerScope** - This tool is a process-level energy consumption measurement tool. But it has a disadvantage, it requires a hardware power meter to measure the energy consumption.

    Requires a hardware power meter to be installed.

16. **Prometheus NE** - Prometheus itself is not able to gather metrics from the hardware directly. However, Prometheus Node Exporter (NE) offers a way to gather metrics from the hardware. It is compatible with OpenStack and Kubernetes. However, it does not measure energy consumption by itself. It needs to be combined with a tool that can measure energy consumption. This means that it is not applicable to solve the problem completely. In addition to that, Prometheus offers a structure for metrics, but it does not offer a way to calculate the carbon footprint.

    Requires combination with an energy approach, but can provide resource metrics at various levels of abstraction.

17. **OpenTelemetry** - OpenTelemetry is an open source observability framework. It is compatible with OpenStack and Kubernetes. It provides a good way to monitor the infrastructure, but it needs to be combined with a model that can calculate the energy consumption and carbon footprint.

    Requires combination with an energy approach, but can provide resource metrics at various levels of abstraction.

[41] https://www.microsoft.com/en-us/research/project/joulemeter-computational-energy-measurement-and-optimization/
[42] https://powerapi.org/reference/sensors/hwpc-sensor/

18. **Venus** - Venus is an OpenStack project. It is more focused on retrieving logs from the infrastructure and not on metrics of running applications. Also, it can only be used in combination with OpenStack and needs administrative rights.

    Not focused on metrics of running applications and requires administrative rights on OpenStack.

19. **Monasca** - Monasca is a promising tool that can obtain CPU and memory utilization. Monasca relies on Keystone for running the service. One of the requirements is that the API must have an admin token. This means that we need administrative rights to install this tool with OpenStack.

    Requires administrative rights on OpenStack.

20. **Ceilometer** - Ceilometer is also a tool designed for OpenStack. It has the same obstacles as Monasca.

    Requires administrative rights on OpenStack.

21. **OProfile** - One downside of OProfile is it can not measure energy consumption at the container level without additional setup. This results in accuracy loss. So, they should be used with caution. Probably, another tool has the preference over this tool.

    Requires combination with an energy approach, but can provide resource metrics at various levels of abstraction.

22. **cAdvisor** - cAdvisor is a tool that can be used to collect system metrics up to container level. It can collect system-wide and on container level, but can interfere with results because it is running in the same container. In Kubernetes, cAdvisor is integrated into the Kubelet binary. It is pretty intelligent to auto-discover all the containers running in the machine, collect CPU, memory, file system, and network usage statistics, and provide a comprehensive overall machine usage by analyzing the root container[43].

    Requires combination with an energy approach, but can provide resource metrics at various levels of abstraction.

3.5 Alternatives

The previous section has provided a comprehensive list of tools along with their minimum requirements for applicability. While constraints may initially limit options, exploring potential solutions can lead to finding the optimal approach. Therefore, we will present four design alternatives aimed at addressing the issues outlined in the case study. These alternatives offer valuable insights for the Digital Lab, particularly when operating with fewer constraints. It is important to note that this section focuses solely on constraints for different alternatives, as the requirements are established by the Digital Lab itself. These requirements will eliminate many tools, especially those incapable of monitoring container-level activity.

The first alternative represents the ideal scenario with all constraints relaxed. In the second alternative, we consider the hardware constraint. The third alternative maintains both the hardware constraint and assumes no access to the hypervisor. Lastly, the fourth alternative integrates all identified constraints and serves as the baseline scenario.

For each alternative, we compile a list of compatible tools. In cases where a single tool falls short of providing all necessary functionalities, we explore combinations with other tools to achieve the desired outcome. Table 5 offers a comprehensive overview of the tools applicable to each alternative.

3.5.1 Alternative 1 - All constraints relaxed

For this first alternative, we will assume that all constraints are relaxed. This means that we have access to the hardware, hypervisor, and administrative rights on OpenStack. Therefore, we can use any tool to collect the

[43] https://scaleyourapp.com/what-is-cadvisor-how-does-it-work-explained/

Tools	Frameworks	Multi-Tenancy	Container Level
Cloud Carbon Footprint (CCF)	x		
Scaphandre	x	x	x
Kepler	x	x	x
Containergy	x	x	x
PAPI	x		
RAPL	x		
DEM	x		
CloudMonitor	x		
Joulemeter	x		
Energy Checker	x		
PowerTOP	x		
Jalen	x		
PowerAPI	x	x	x
pTop	x		
PowerScope	x		
Prometheus NE	x	x	x
OpenTelemetry	x	x	x
Venus	x	x	
Monasca	x	x	
Ceilometer	x	x	
OProfile	x		
cAdvisor	x	x	x

Table 3: Technology Assessment: Requirements

data. This alternative will be the ideal solution. If we take a look at Table 4 we can use all tools that have a cross or tilde in all the container-level columns and meet all the requirements. However, since this alternative has the same applicable tools as the second alternative, we will continue with the second alternative.

3.5.2 Alternative 2 - No access to hardware

Most tools that measure on the container level do not need access to the hardware physically. This means that we can use all of the tools that apply to the first alternative. If this alternative is chosen the preference is to use tools that can monitor energy out-of-the-box (OOTB). Therefore, some of the tools below, that need the support of an Energy Model (EM), will not be used for this alternative. The following tools are therefore applicable to this alternative:

- Scaphandre
- Kepler
- Containergy
- PowerAPI
- Prometheus Node Exporter + EM

Category	Tool	Hardware				Admin Rights OpenStack				Hypervisor access			
		BM	VM	CLU	CON	BM	VM	CLU	CON	BM	VM	CLU	CON
Footprint	CCF	x				x				x	x		
Energy	Scaphandre	x	x	x	x	x	x	x	x	∼	∼	∼	∼
	Kepler	x	x	x	x	x	x	x	x	x	x	x	x
	Containergy				x				x				∼
	PAPI	x				∼				x	x		
	RAPL	x				∼				x	x		
	DEM	x	x	x		x	x	x		x	x	x	
	CloudMonitor	x				∼				x	x		
	Joulemeter	x				x				x			
	Energy Checker	∼				x				x			
	PowerTOP	x				x				x	x		
	Jalen	x				x				x			
	PowerAPI	x	x	x	x	x	x	x	x	∼	∼	∼	∼
	pTop	x				x				x			
	PowerScope	∼				x				x			
Metrics	Prometheus NE			x	x			x	x	∼	x	x	x
	OpenTelemetry HMR			x	x			x	x	∼	x		x
	Venus	x	x			∼	∼			x	x		
	Monasca	x	x			∼	∼			x	x		
	Ceilometer	x	x			∼	∼			x	x		
	OProfile	x	x			x	x			x	x		
	cAdvisor	x	x	x	x	x	x	x	x	x	x	x	x

Table 4: Technology Assessment: Constraints
x: Can measure with the constraint
∼: If no constraint it can measure on this level

- OpenTelemetry Host Metrics Receiver + EM
- cAdvisor + EM

3.5.3 Alternative 3 - No access to hardware nor hypervisor

For this alternative, we will assume that we do not have access to the hardware and hypervisor. This means that, in addition to the previous alternative, we can not use any tool that measures with the help of the hypervisor. Since this alternative has the same applicable tools as the fourth alternative, we will continue with the next alternative.

3.5.4 Alternative 4 - All constraints considered

This final alternative will consider all the constraints and requirements. In comparison to the previous alternative, it adds administrative rights to OpenStack. However, this extra constraint does not lock any tool out. This means that we can use the following tools below. As we can see, all of the tools need the support of an Energy Model (EM) to translate utilization metrics into energy consumption.

Name	Last Update	Energy OOTB	Alternative	BM	VM	CLU	CON
Scaphandre	2023	Yes	2	x	x	x	x
Kepler	2023	Yes	2	x	x	x	x
			4	x	x	x	x
Containergy	2020	Yes	2	-	-	-	x
PowerAPI	2023	Yes	2	x	x	x	x
Prometheus NE + EM	2023	No	2	x	x	x	x
			4	-	x	x	x
OpenTelemetry HMR + EM	2023	No	2	x	x	-	x
			4	-	x	-	x
cAdvisor + EM	2023	No	2	x	x	x	x
			4	x	x	x	x

Table 5: Alternatives Overview

- Kepler (Existing Pre-trained model)
- Prometheus Node Exporter + EM
- OpenTelemetry Host Metrics Receiver + EM
- cAdvisor + EM

3.6 Setup

Before proceeding to the next sections on exploratory tool testing and implementation, it is important to outline the setup along with all the known specifications used for all explorations and implementations.

We utilize servers owned by Digital Lab and hosted in the University of Groningen's data center. We have been allocated some resources to use for the experiments and implementations. We are interfacing with an individual bare metal server that possesses the following specifications:

- **CPU:** Dual AMD EPYC™ 7763 processor, totaling 128 physical cores with 2 virtual CPUs per physical core, resulting in a total of 256 vCPUs/Threads.
- **Memory:** 512GB RAM

We have set up three virtual machines (VMs) on this server to deploy a Kubernetes cluster. The VMs are configured with the following specifications, all running Ubuntu ARM server images:

1. **master-node:** 3 vCPUs, 15.6GB RAM, 20GB disk.
2. **worker-node1:** 2 vCPUs, 8GB RAM, 10GB disk.
3. **worker-node2:** 2 vCPUs, 4GB RAM, 10GB disk.

Before proceeding, it is practical to gather more information about each VM, as they may be deployed on different underlying hardware. To achieve this, the following commands are executed on each node:

- **Linux Kernel version:** `uname -r`

Name	Type	Description
system_cpu_load_average_nm_ratio	gauge	The ratio of the system load average to the number of logical CPUs.
system_cpu_time_seconds_total	counter	Total user and system CPU time spent in seconds.
system_cpu_utilization_ratio	gauge	The ratio of the CPU utilization.
system_memory_usage_bytes	gauge	The number of bytes used in memory.
system_memory_utilization_ratio	gauge	The ratio of the memory utilization.

Table 6: Host Metrics Receiver metrics

1. **master-node:** 5.4.0-137-generic
2. **worker-node1:** 5.4.0-137-generic
3. **worker-node2:** 5.4.0-137-generic

- **CPU details:** `lscpu`

1. **master-node:** AMD EPYC-Milan Processor x86_64
2. **worker-node1:** AMD EPYC-Milan Processor x86_64
3. **worker-node2:** AMD EPYC-Milan Processor x86_64

As we observe, the output is consistent across all nodes, allowing us to confidently proceed.

3.7 Exploratory Tool Testing

The assessment of technology and alternatives relies solely on the documentation provided by the tools. However, at times, we encountered uncertainties in the documentation, which lowered our confidence in the tools' applicability. Additionally, there is a possibility that the documentation might not be completely up-to-date. Consequently, we felt it necessary to conduct experiments with some of the tools to validate if they indeed meet the requirements outlined in the documentation. To address these uncertainties, we proceeded with our experiments as outlined below. Our primary focus will be on alternative four, as it will serve as our baseline scenario.

3.7.1 Scaphandre

There were concerns regarding the hardware compatibility of Scaphandre. The documentation showed some doubt about the compatibility of CPUs and whether they can be deployed on a virtual machine. Considering that Digital Lab plans to implement OpenTelemetry, we opted to experiment with the Host Metrics Receiver plugin of this tool. Therefore, we conducted a quick test to determine if it is compatible with the machines that we use.

The virtual machines we employ for the experiment, which will also be utilized in the final production environment of Digital Lab, are based on AMD processors. This poses a challenge for using Scaphandre because it relies on IntelRAPL under the hood. We can verify the availability of IntelRAPL by running the following command: `egrep "(pts|pln)" /proc/cpuinfo`. This command returns nothing, indicating that IntelRAPL is not available, which is also confirmed by the Scaphandre documentation.

The compatibility page of Scaphandre[44] states that it can be used for AMD x86 bare metal servers. However, it requires a Linux kernel of version 5.11 or higher, which is not the case for the machines we use.

[44] https://hubblo-org.github.io/scaphandre-documentation/compatibility.html

Name	Type	Description
node_cpu_seconds_total	counter	Seconds the cpus spent in each mode.
node_memory_MemTotal_bytes	gauge	Total amount of memory.
node_memory_MemFree_bytes	gauge	The amount of free memory.
node_memory_Cached_bytes	gauge	The amount of cached memory.
node_memory_Buffers_bytes	gauge	The amount of buffered memory.
node_memory_SReclaimable_bytes	gauge	The amount of reclaimable memory.
container_cpu_usage_seconds_total	counter	Cumulative CPU usage in seconds.
container_memory_usage_bytes	gauge	Current memory usage in bytes.

Table 7: Prometheus Node Exporter metrics

Despite the warnings, we decided to try it ourselves. Therefore, we installed the helm chart and deployed it on all the nodes in the cluster. However, we encountered an instant `CrashLoopBackOff` and eventually an `Error` status. The following logs confirm the incompatibility, as also stated in the documentation. `scaphandre:: sensors:: powercap_rapl: Couldn't find intel_rapl modules. scaphandre:: sensors:: powercap_rapl: Couldn't find domain folders from powercap. Fallback on socket folders. scaphandre:: sensors:: powercap_rapl: Scaphandre will not be able to provide per-domain data.`

This marks the end of our experimentation with Scaphandre. Consequently, we cannot use Scaphandre for our specific use case.

3.7.2 Kepler

For alternative four Kepler looks like a good candidate. As already discussed in the previous sections, Kepler needs a pre-trained model to estimate the energy consumption at VM level. Diving into the documentation[45] we found that the current existing model is developed in `Intel® Xeon® Processor E5-2667 v3`. They mention that models with other architectures will be coming soon. However, this means that the current existing pre-trained model is not compatible with the AMD EPYC 7763 processor that we use in our setup. Therefore, we decided not to experiment with Kepler.

3.7.3 OpenTelemetry Host Metrics Receiver

We utilize the OpenTelemetry demo project for this tool, which can be set up on a Kubernetes cluster using the provided Helm chart[46]. The demo showcases a microservice-based application written in various programming languages that interact with each other, along with a load generator simulating traffic.

With the demo, we can install the Host Metrics Receiver plugin of OpenTelemetry. Enabling the `daemonset` for the OpenTelemetry collector is crucial as it establishes a daemon on every node in the cluster, allowing us to retrieve metrics indicating where a process is running. Subsequently, we activate the plugin itself to assess whether the Host Metrics Receiver exports the relevant data while operating within our setup.

Upon observation, it is evident that the Host Metrics Receiver successfully exports the desired data, including metrics related to CPU, memory, disk, and network. Some of these metrics, related to VM-level estimations, are detailed in Table 6. However, despite these metrics, there is a notable absence of data at the container level. Consequently, we explore Prometheus Node Exporter as an alternative avenue.

[45] `https://sustainable-computing.io/design/power_model/`

[46] `https://opentelemetry.io/docs/demo/kubernetes-deployment/`

Name	Type	Description
cpuDescription	string	Central processing unit description
memory	number	Number of gigabytes of memory usage
machineType	string	Machine type (Ie. server, laptop, desktop)
startTime	Date	Timestamp recording the start day/time of usage
endTime	Date	Timestamp recording end day/time of usage
country	string	Country where server was located
region	string	Region or state within country server was located
machineName?	string	Physical host name
cost?	number	The amount of cost associated with each row
cpuUtilization?	number	Specific server utilization percentage (Ie. 50% = 50)
powerUsageEffectiveness?	number	Power usage effectiveness for data center
dailyUptime	number	Active usage hours in the last day
weeklyUptime	number	Active usage hours in the last week
monthlyUptime	number	Active usage hours in the last month
annualUptime	number	Active usage hours in the last year

Table 8: Cloud Carbon Footprint On-Premise Input

3.7.4 Prometheus Node Exporter

Prometheus Node Exporter is a handy tool for exporting host metrics. We decided to experiment with this tool because the OpenTelemetry Host Metrics Receiver does not scrape metrics at the container level. Hence, our focus was particularly on finding container metrics using this tool.

After conducting several tests, we found that Prometheus Node Exporter effectively scrapes the metrics we need, which are crucial for monitoring container utilization. The metrics we are interested in are listed in Table 7. Notably, this tool can gather metrics from both virtual machine and container levels, offering a significant advantage as it merges our monitoring needs into one tool. It achieves this by leveraging cAdvisor, as outlined in a Medium article[47]. Essentially, Prometheus Node Exporter serves as the scraper for various levels of abstraction, while cAdvisor retrieves the metrics. cAdvisor, in turn, interacts with Kubernetes Kubelet to access metrics from within the cluster.

3.7.5 Cloud Carbon Footprint

Since none of the available energy tools are compatible with alternative four, we must seek an energy model that aligns with a system host metric tool. The Cloud Carbon Footprint (CCF) tool is a good candidate for this, but we first want to experiment with it to see if it meets our requirements.

CCF is designed to estimate energy usage and carbon emissions within cloud infrastructure. Our goal is to assess if it can be used to make estimations with only host metrics scraped from the Kubernetes cluster.

Our investigation starts with installing the tool locally on our machine. The installation process is facilitated through `npx`, providing a user-friendly setup. With the tool set up, we proceed to input data. Presently, CCF solely supports data input and output via CSV files for on-premise measurements. Table 8 lists the required data points for estimation, with some being optional. Following data input, we execute the command `yarn run estimate-on-premise-data --onPremiseInput [<Input File Name>]`, which generates an output CSV file detailing kilowatt hours and CO2e for each uptime increment.

[47] `https://medium.com/@onai.rotich/understand-container-metrics-and-why-they-matter-9e88434ca62a`

In our exploration, we examined the `cpuDescription` field's impact on estimation accuracy. Initially, we observed seemingly consistent results regardless of the input. However, upon closer inspection, we discovered the necessity of specifying the exact CPU description. For instance, identifying our CPU as `AMD EPYC 7763` without appending `2.25Ghz` yielded accurate estimations.

4 Design & Implementation

This section will present a model and implementation to address the problem outlined in the case study and achieve the stated objective. Initially, in Section 4.1, we will outline the chosen technology stack. Following that, in Section 4.2, we will introduce the model employed to estimate energy consumption and carbon footprint, relying on utilization metrics. All relevant equations will be covered within this section. Finally, the implementation is outlined in Section 4.3. This section consists of two subsections. First, Section 4.3.1 will detail how the model of Section 4.2 is implemented. Second, Section 4.3.2 will show how the dashboard is implemented. This dashboard will visualize the results of the implementation.

4.1 Design Decisions

Given the alternatives and experimentation, we have arrived at several key design choices. Our decision to utilize existing tools is because they are proven and have a large community. By doing so, we ensure access to reliable support and ongoing updates. Thus, we the implementation shows to be stable and accurate we enhance the longevity of our project.

- **Prometheus Node Exporter**
  The main reason we selected Prometheus Node Exporter is that it demonstrated its ability to monitor container-level abstraction by using cAdvisor under the hood. On top of that, it can auto-detect running containers and monitor their resource usage directly. This is a key feature for our project, as it allows us to monitor the resource usage of containers without having to manually configure each one. It offers host metrics such as CPU and memory utilization.

- **Cloud Carbon Footprint**
  Resource utilization metrics need to be paired with an energy estimation model. Several models exist, but Cloud Carbon Footprint (CCF) has emerged as a promising tool. It is an open-source project with significant contributions, making it more than just a standalone energy equation it is a project with strong community involvement. As of the time of writing this thesis (March 2024), CCF has garnered 817 stars on GitHub[48] and continues to receive regular contributions each month. Furthermore, the project offers both energy and carbon emissions estimations simultaneously, enhancing its utility.

- **Electricity Maps**
  We selected Electricity Maps for our carbon intensity data. Among the tools we reviewed, all relied on ENTSO-E as their data source. We opted for Electricity Maps due to its user-friendly interface. Additionally, Electricity Maps has merged with CO2 Signal, indicating its growth. Furthermore, CCF utilizes Electricity Maps as their data source for their non-on-premises approaches.

- **Grafana Dashboard**
  Grafana[49] is widely used for displaying time series data, making it a suitable tool for our project. Its compatibility with Prometheus via PromQL makes it a strong choice. Additionally, its advanced visualization features are beneficial. This aspect is particularly crucial for our project as we aim to visualize energy and carbon footprint data across various levels of abstraction.

4.2 Model Design

This thesis presents an extension method on the CCF model to estimate energy usage and carbon emissions at the BM level, based exclusively on metrics from virtual machines. Furthermore, it distributes the calculated energy consumption and associated emissions among virtual machines and containers.

[48] https://github.com/cloud-carbon-footprint/cloud-carbon-footprint
[49] https://grafana.com

We first define the symbols used in the equations in Section 4.2.1. Next, we explain how we prepare the virtual machine metrics so that they can be given to the CCF model in Section 4.2.2. Section 4.2.3 then employs these prepared metrics as input for the equations originating from the CCF main methodology. Finally, Sections 4.2.4 and 4.2.5 address the distribution of energy and carbon footprint from the bare metal server to the virtual machines and containers. The sections labeled with `Extension` are the new parts that we have added to the CCF model.

The steps in the following sections have to be repeated for a certain interval. The time window can be chosen freely. It is recommended to use the same interval as the Prometheus scrape interval for virtual machine metrics. This ensures that the metrics are collected at the same time and are consistent with each other. On top of that, when having a smaller interval the below steps are calculated for the same metric values, which in turn cause overhead.

4.2.1 Symbols

To improve the readability of the equations below, we have introduced the following symbols:

E: Energy in kWh

$CO2eq$: Carbon Dioxide Equivalent in mg

K: Known metric values (e.g. CPU utilization, memory usage, etc.)

UK: Unknown metric values

BM: Bare Metal

VM: Virtual Machine

CON: Container

MU: Memory Usage in GiB

MT: Memory Total in GiB

U: CPU Utilization in %

$vCPU$: Virtual CPU

$vCPUs$: List of vCPUs

$pCPU$: Physical CPU

Sum: Sum of the values. E.g. sum of all CPU utilization percentages

$Median$: Median of the values in a list

4.2.2 CCF Model Extension: Metric Preparation

The CCF model depends on certain metrics outlined in Table 8, with a particular focus on CPU and memory utilization. Consequently, this section is dedicated to gathering these metrics. Three approaches are being considered:

1. Obtaining hypervisor access would provide precise data regarding physical CPU, total memory usage, vCPUs information, and overhead. However, this option is often not feasible as access is typically restricted to the virtual machine.

2. Assuming equal CPU and memory utilization between the virtual and physical machines, ignoring the difference in abstraction levels.

3. Try to find how many virtual CPUs and the total memory are on a physical CPU, so we can apply the average or median of the known virtual machine metrics to the unknown virtual machines.

When opting for the first or second approach, this section may be omitted as the necessary input for the following section is already accessible. The CPU and memory utilization metrics can then be directly provided to the CCF model. If there are restrictions on hypervisor access, only the second or third approach is appropriate. Considering that the third approach is likely to provide values closer to the actual ones compared to the second approach, we outline the third approach in this section.

Below, we outline the method for this third approach. First, we calculate the median or average of the known virtual machine metrics. Subsequently, these computed values are then applied to the unknown virtual machines. Finally, we derive an average of the metrics from both the known and unknown virtual machines to estimate the metrics of the physical machine.

1. Sum of all known vCPUs values
   Equation 6 calculates the sum of all known vCPUs values. This is done by summing the CPU utilization of each vCPU.
$$SumU_{K_{VM}} = \sum_{vCPU \in vCPUs} U_{K_{vCPU}} \tag{6}$$

2. Median or Fifty Percent methodology
   Equation 7 provides a variable that is used to determine if the median or fifty percent methodology should be used. If the median is enabled, the median of the known vCPUs utilization values is used. Otherwise, the value is set to 0.5. This value will be used to indicate that fifty percent of the vCPU is utilized. The CCF model uses 50% as the default value for the average server utilization[50]. CCF based this value on the projected estimate for the average server utilization of servers in hyperscale data centers in 2020 of 50% [35].
$$medianOrFiftyPercent = \begin{cases} MedianU_{K_{VM}}, & \text{if } MEDIAN\_ENABLED = True \\ 0.5 & \text{otherwise} \end{cases} \tag{7}$$

3. Sum of all unknown vCPUs values
   Equation 8 calculates the sum of all unknown vCPUs utilization values. This is done by multiplying the difference between the total number of vCPUs on the bare metal server and the total number of vCPUs on the known virtual machines with variable medianOrFiftyPercent from Equation 7.
$$SumU_{UK_{VM}} = (k-n) \times medianOrFiftyPercent \tag{8}$$
   where
   - k is the total number of vCPUs on the bare metal server
   - n is the total number of vCPUs on the known virtual machines

4. Sum of all vCPUs values of the Bare Metal Server
   Equation 9 calculates the sum of all vCPUs values of the bare metal server. This is done by summing the sum of all known vCPUs values and the sum of all unknown vCPUs values.
$$SumU_{BM} = SumU_{K_{VM}} + SumU_{UK_{VM}} \tag{9}$$

[50] https://www.cloudcarbonfootprint.org/docs/on-premise/#methodology

5. CPU Utilization of the Bare Metal Server
   Equation 10 calculates the CPU utilization of the bare metal server. This is done by dividing the sum of all known and unknown vCPUs values by the total number of vCPUs on the bare metal server. With this, we get the average CPU utilization of the bare metal server based on the known and unknown virtual machines.

$$U_{BM} = \frac{SumU_{BM}}{k} \tag{10}$$

   where

   - k is the total number of vCPUs on the bare metal server

6. Sum of all known memory usage values
   Equation 11 calculates the sum of memory usage of each known virtual machine.

$$SumMU_{K_{VM}} = \sum_{VM \in VMs} MU_{K_{VM}} \tag{11}$$

7. Sum of all memory total values
   Equation 12 calculates the sum of total memory available on each known virtual machine.

$$SumMT_{K_{VM}} = \sum_{VM \in VMs} MT_{K_{VM}} \tag{12}$$

8. Memory Usage Percentage of known virtual machines
   Equation 13 calculates the memory usage percentage of the known virtual machines. This is done by dividing the sum of memory usage of the known virtual machines from Equation 11 by the sum of the total memory of the known virtual machines from Equation 12.

$$PercentageMU_{K_{VM}} = \frac{SumMU_{K_{VM}}}{SumMT_{K_{VM}}} \tag{13}$$

9. Memory Total of unknown Virtual Machines
   Equation 14 calculates the total memory of the unknown virtual machines. This is done by subtracting the sum of total memory of the known virtual machines from Equation 12 from the total memory of the bare metal server.

$$MT_{UK_{VM}} = MT_{BM} - SumMT_{K_{VM}} \tag{14}$$

   where

   - MT_{BM} is the total memory of the bare metal server that is user input

10. Memory Usage of the Unknown Virtual Machines
    Equation 15 calculates the memory usage of the unknown virtual machines. This equation has two cases.

    The first case is when the average memory is enabled, then the memory usage percentage of the known virtual machines is multiplied by the total memory of the unknown virtual machines. This gives us the memory usage of the unknown virtual machines. We use an average calculation instead of a median because we are unable to calculate the median memory of the unknown virtual machines. For CPU utilization, we do know the total number of vCPUs so we can calculate the median. However, for memory, we do not know how the unknown memory is distributed across how many virtual machines or vCPUs. Therefore, we use the average.

    The second case is when the average memory is not enabled, then the total memory of the unknown virtual machines is divided by two. Both cases provide the memory usage of the unknown virtual machines.

$$MU_{UK_{VM}} = \begin{cases} PercentageMU_{K_{VM}} \times MT_{\mathrm{UK_{VM}}}, & \text{if } MEM\_AVG = True \\ \frac{MT_{\mathrm{UK_{VM}}}}{2}, & \text{otherwise} \end{cases} \tag{15}$$

11. Memory Usage of the Bare Metal Server
Equation 16 calculates the memory usage of the bare metal server. This is done by summing the memory usage of the known virtual machines from Equation 11 and the memory usage of the unknown virtual machines from Equation 15.

$$MU_{BM} = SumMU_{K_{VM}} + MU_{UK_{VM}} \tag{16}$$

4.2.3 CCF Model: Energy Consumption and Carbon Footprint of BM

This section applies the Cloud Carbon Footprint main methodology to estimate the energy consumption and carbon footprint of the bare metal server.

1. Average Watts to be used
Equation 17 calculates the average watts value. There are two cases to consider. The first case is when the average watts are already known and CPU utilization is not considered. The second case is when the average watts are not known and the minimum and maximum watts are known. In this case, the average watts are calculated by adding the minimum watts to the product of the CPU utilization factor and the difference between the maximum and minimum watts.

$$AvgWatts = \begin{cases} \text{averageWatts}, & \text{if averageWatts} \neq 0 \\ minWatts + U_{BM} \times (maxWatts - minWatts), & \text{otherwise} \end{cases} \tag{17}$$

where

- *averageWatts* is the average watts of the bare metal server (user input)
- *minWatts* is the minimum watts of the bare metal server (user input)
- *maxWatts* is the maximum watts of the bare metal server (user input)

2. Energy in kWh of the CPU Utilization
Equation 18 calculates the energy in kWh of the CPU utilization. This is done by multiplying the average watts we calculated in Equation 17 with the uptime usage of the CPU in hours, the Power Usage Effectiveness (PUE) of the data center, and the Replication factor. These factors are introduced in Section 2.2. Keep in mind that this amount of energy only considers the CPU utilization.

$$E_{U_{BM}} = \frac{AvgWatts \times H \times PUE \times RF}{1000} \tag{18}$$

where

- H uptime usage of the CPU in hours
- PUE is the Power Usage Effectiveness of the data center
- RF is the Replication factor
- E is the energy in kWh

3. Carbon Dioxide Equivalent in metric tons of the CPU Utilization
Equation 19 calculates the carbon dioxide equivalent in metric tons of the CPU utilization. This is done by multiplying the energy in kWh of the CPU utilization with the carbon intensity of the electricity.

$$mtCO2eq_{U_{BM}} = E_{U_{BM}} \times c \tag{19}$$

where

- c is the carbon intensity of the electricity in mtCO2eq/kWh

4. GbHours of the Memory Usage
   Equation 20 calculates the gigabyte hours of memory usage. This is done by multiplying the memory usage with the uptime. Memory usage is the max of either zero or the usage of memory minus the average processor memory from the SPECPower database. The uptime refers to the duration, measured in hours, during which the server has been operational.

$$GbHours = \max(0, MU_{BM} - MEM\_AVG) \times H \tag{20}$$

   where

   - MEM_AVG is the average memory that is used in the SPECPower benchmark[51] for this processor.
   - H is the uptime of the server in hours. In this case, calculated in time windows of 15 seconds

   The max function is used to ensure that we do not estimate a double amount of energy usage. The CCF model only applies additional estimation when our bare metal server has a higher amount of memory usage than what is provided in the SPECPower database. This means that the CPU utilization is responsible for the energy usage of the memory. When using the watts from the SPECPower database, the energy usage of the memory is already included in the CPU utilization. Therefore, Equation 20 will only result in another value than zero when the actual memory usage is higher than the total memory used in the SPECPower database for this system architecture.

5. Energy in kWh of the Memory Usage
   Equation 21 calculates the energy in kWh of the memory usage. This is done by multiplying the gigabyte hours from Equation 20 with the memory coefficient, the PUE of the data center, and the Replication factor. The memory coefficient is the average of the power usage of DDR4 and DDR5 memory.

   The usage of DDR4 or DDR5 memory is assumed. Manufacturers of these memories offer a rule of thumb. Crucial[52] suggests allocating approximately 3 watts of power for every 8GB, which amounts to `0.375 W/GB`. However, Micron[53] states that each DRAM will consume approximately `408.3mWh` of total power, which amounts to `0.4083 W/GB`. CCF decided to use the average of both values, which is `0.392 W/GB`. Thus, the application uses `0.000392 kWh/GbHours`.

$$E_{MU_{BM}} = GbHours \times mc \times PUE \times RF \tag{21}$$

   where

   - mc is the memory coefficient. This is 0.000392 kWh/GbHours. This is the average of the power usage of DDR4 and DDR5 memory.
   - PUE is the Power Usage Effectiveness of the data center
   - RF is the Replication factor This replication factor is default set to only one.
   - E is the energy in kWh

6. CO2eq in metric tons of the Memory Usage
   Equation 22 calculates the carbon dioxide equivalent in metric tons of memory usage. This is done by multiplying the energy in kWh of the memory usage with the carbon intensity of the electricity.

$$mtCO2eq_{MU_{BM}} = E_{MU_{BM}} \times c \tag{22}$$

   where

[51] `https://www.spec.org/power_ssj2008/`

[52] `https://www.crucial.com/support/articles-faq-memory/how-much-power-does-memory-use`

[53] `https://www.micron.com/-/media/client/global/documents/products/technical-note/dram/tn4007_ddr4_power_calculation.pdf`

- c is the carbon intensity of the electricity in mtCO2eq/kWh

7. Energy in kWh of the Bare Metal Server
Equation 23 calculates the energy in kWh of the bare metal server. This is done by summing the energy in kWh of the CPU utilization from Equation 18 and the energy in kWh of the memory usage from Equation 21.

$$E_{BM} = E_{U_{BM}} + E_{MU_{BM}} \tag{23}$$

8. CO2eq in metric tons of the Bare Metal Server
Equation 24 calculates the carbon dioxide equivalent in metric tons of the bare metal server. This is done by summing the carbon dioxide equivalent in metric tons of the CPU utilization from Equation 19 and the carbon dioxide equivalent in metric tons of the memory usage from Equation 22.

$$mtCO2eq_{BM} = mtCO2eq_{U_{BM}} + mtCO2eq_{MU_{BM}} \tag{24}$$

4.2.4 CCF Model Extension: Share VM of BM

1. Share of CPU Utilization of the Virtual Machine to the Bare Metal Server
Equation 25 calculates the share of CPU utilization of the virtual machine to the bare metal server. This is done by dividing the CPU utilization of an individual virtual machine by the sum of all CPU utilization of the bare metal server from Equation 9.

$$UShareBM_{VM} = \frac{U_{VM}}{SumU_{BM}} \tag{25}$$

2. Share of Memory Usage of the Virtual Machine to the Bare Metal Server
Equation 26 calculates the share of memory usage of the virtual machine to the bare metal server. This is done by dividing the memory usage of an individual virtual machine by the memory usage of the bare metal server from Equation 16.

$$MShareBM_{VM} = \frac{MU_{VM}}{MU_{BM}} \tag{26}$$

3. Share of the Virtual Machine to the Bare Metal Server
Equation 27 calculates the share of the virtual machine to the bare metal server. This is done by taking the maximum share of either the CPU utilization from Equation 25 or the memory usage from Equation 26.

$$ShareBM_{VM} = \max(UShareBM_{VM}, MShareBM_{VM}) \tag{27}$$

4. Energy in kWh of the Virtual Machine
Equation 28 calculates the energy in kWh of an individual virtual machine. This is done by multiplying the share of the virtual machine to the bare metal server from Equation 27 with the energy in kWh of the bare metal server from Equation 23.

$$E_{VM} = ShareBM_{VM} \times E_{BM} \tag{28}$$

5. CO2eq in metric tons of the Virtual Machine
Equation 29 calculates the carbon dioxide equivalent in metric tons of an individual virtual machine. This is done by multiplying the share of the virtual machine to the bare metal server from Equation 27 with the carbon dioxide equivalent in metric tons of the bare metal server from Equation 24.

$$mtCO2eq_{VM} = ShareBM_{VM} \times mtCO2eq_{BM} \tag{29}$$

4.2.5 CCF Model Extension: Share CON of VM

1. Share of CPU Utilization of the Container to the Virtual Machine
   Equation 30 calculates the share of CPU utilization of the container to the virtual machine. This is done by dividing the CPU utilization of the container by the CPU utilization of the virtual machine.

$$UShareVM_{CON} = \frac{U_{CON}}{U_{VM}} \tag{30}$$

2. Share of Memory Usage of the Container to the Virtual Machine
   Equation 31 calculates the share of memory usage of the container to the virtual machine. This is done by dividing the memory usage of the container by the memory usage of the virtual machine.

$$MShareVM_{CON} = \frac{MU_{CON}}{MU_{VM}} \tag{31}$$

3. Share of the Container to the Virtual Machine
   Equation 32 calculates the share of the container to the virtual machine. This is done by taking the maximum share of either the CPU utilization from Equation 30 or the memory usage from Equation 31.

$$ShareVM_{CON} = \max(UShareVM_{CON}, MShareVM_{CON}) \tag{32}$$

4. Energy in kWh of the Container
   Equation 33 calculates the energy in kWh of the container. This is done by multiplying the share of the container to the virtual machine from Equation 32 with the energy in kWh of the virtual machine from Equation 28.

$$E_{CON} = ShareVM_{CON} \times E_{VM} \tag{33}$$

5. Carbon Dioxide Equivalent in metric tons of the Container
   Equation 34 calculates the carbon dioxide equivalent in metric tons of the container. This is done by multiplying the share of the container to the virtual machine from Equation 32 with the carbon dioxide equivalent in metric tons of the virtual machine from Equation 29.

$$mtCO2eq_{CON} = ShareVM_{CON} \times mtCO2eq_{VM} \tag{34}$$

4.3 Implementation

Given that we have a model design and have chosen the relevant tools that can support this model, we can proceed with the implementation. The implementation is divided into two parts. The first part involves implementing the model and supportive tools, as discussed in Section 4.2. The second part focuses on implementing the dashboard, as detailed in Section 4.3.2.

4.3.1 Model Implementation

The model as described in Section 4.2.2 can be utilized in three different ways. Since we know the total number of vCPUs and the total amount of memory of the bare metal we can use option three. This means that we need to collect all known vCPU utilization values and all known memory usage values.

1. **Input Parameters**
   First, we outline the input parameters for our implementation, which align with the setup detailed in Section 3.6. Therefore we know our bare metal hardware specifications.

   Based on the `SPECpower_ssj2008` results, we derive the average memory from all entries sharing the same CPU description. However, to determine the average minimum and maximum power consumption,

we consider records with both the same CPU description and thread count. This is crucial as the number of threads significantly influences power usage. See Table 9 for all records with the same CPU description. Although our specific configuration is not directly matched, we utilize records with identical thread counts, as the model incorporates kWh per GB of additional memory usage.

vCPUs: 256

Memory: 512GB

CPU Description: AMD EPYC 7763

Average SPECpower_ssj2008 Memory: 332.8 GB

Min. Watts: 98.2 watts

Max. Watts: 455.8 watts

Date	MHz	Chips	Cores	Threads	Memory	min. watts	max. watts
15-3-2021	2450	1	64	128	128	58	272
15-3-2021	2450	1	64	128	128	58	268
15-3-2021	2450	1	64	128	128	53.7	270
30-6-2021	2450	1	64	128	128	53	228
30-6-2021	2450	1	64	128	128	52.5	228
30-6-2021	2450	1	64	128	128	54	240
13-7-2022	2450	2	128	256	256	94.6	408
13-7-2022	2450	2	128	256	256	122	406
15-3-2021	2450	2	128	256	256	120	546
24-3-2021	2450	2	128	256	256	75.6	460
24-3-2021	2450	2	128	256	256	104	459
8-4-2021	2450	2	128	256	256	79.3	454
6-5-2021	2450	2	128	256	256	108	455
6-5-2021	2450	2	128	256	256	102	461
28-10-2021	2450	2	128	256	256	78.5	453
15-3-2021	2450	4	256	512	512	331	1192
15-3-2021	2450	4	256	512	512	257	1173
15-3-2021	2450	6	384	768	768	430	1731
15-3-2021	2450	6	384	768	768	352	1694
15-3-2021	2450	8	512	1024	1024	523	2260
Average					**332.8**	**159**	**695.3**

Table 9: SPECpower_ssj2008 Results for the AMD EPYC 7763

2. **Scrape Data**
   We employ the Prometheus Node Exporter to gather metrics. The Prometheus server retrieves these metrics from the Node Exporter, utilizing the Kubelet API to access metrics at varying levels of abstraction. Given that the CCF model focuses solely on CPU and memory, our interest only lies in the metrics outlined in Table 7. Subsequently, these metrics are stored in persistent volume storage. This practice enables us to utilize the data for the CCF model and display it on the dashboard.

3. **Query Node Metrics**
   After obtaining the metrics, we can begin querying the data from the nodes. The rate function will serve to handle more queries. This function computes the CPU usage rate per second for all vCPUs of each node within the last minute. The memory usage of each node is computed by subtracting the free, cached, and buffered memory from the total memory available. Finally, the total amount of memory is retrieved. The following queries are used:

   - `sum by(node, cpu) (rate(node_cpu_seconds_totalmode!="idle"[1m]))`
   - `node_memory_MemTotal_bytes - node_memory_MemFree_bytes - (node_memory_Cached_bytes + node_memory_Buffers_bytes + node_memory_SReclaimable_bytes)`
   - `node_memory_MemTotal_bytes`

4. **Initialize Variables**
   Before we start with the loop we first initialize some variables. The variables with capital letters are constants and the other variables are updated throughout the loop. The following variables are initialized:

   - `CPU_MEDIAN_ENABLED`
     This boolean is used to determine if we want to use the median or the `AVERAGE_VCPU_UTILIZATION` of the known vCPU utilization
   - `MEMORY_AVG_OF_KNOWN_NODES_ENABLED`
     This boolean is used to determine if we want to use the average of the known nodes memory usage or half of all the unknown memory.
   - `BARE_METAL_TOTAL_VCPU`
     This represents the overall count of vCPUs on the bare metal and this value remains constant. It is determined by multiplying the quantity of physical cores by the count of vCPUs per physical core. In our scenario, we possess 2x AMD EPYC™ 7763, equaling 128 physical cores. The number of vCPUs per physical core is flexible in OpenStack through overcommitment configuration. In our setup, we have set it at 2 vCPUs per physical core, resulting in a total of 256 vCPUs.
   - `AVERAGE_VCPU_UTILIZATION`
     This is the average vCPU utilization. This is a constant and is set to `0.5`. This is used later to assume a vCPU utilization is using 50% its capacity when `CPU_MEDIAN_ENABLED` is set to false.
   - `nodes`
     We initialize an empty array called nodes where we store metric values for each node. This makes it easier to access later.
   - `bareMetalAmountOfUnkownVCPU`
     This variable is used to store the number of vCPUs that are unknown and not used by our setup. We query the vCPU utilization grouped by node and vCPU. This means that we get the vCPU utilization for each vCPU on each node. Therefore, every time we loop over a vCPU we subtract one from the `bareMetalAmountOfUnkownVCPU`.
   - `bareMetalTotalKnownCPUUtilization`
     This variable is used to sum the total vCPU utilization values of all nodes.
   - `BARE_METAL_TOTAL_MEMORY_BYTES`
     This is a constant value of the total memory of the bare metal in bytes. We know from our hardware specifications that the server of Digital Lab has 512GB of memory. This is `549755813888 bytes`.
   - `nodesMemoryUsageBytes`
     This variable is set to zero. When looping through the nodes, the memory usage of each node is added to this variable.

- `nodesMemoryTotalBytes`
  This is the same as the previous one but for the total memory of each node.

5. **Loop through vCPU of nodes for CPU metrics**
   Next, we loop through the result of the node CPU query. First, we add the value of the vCPU utilization to the `bareMetalTotalKnownCPUUtilization`. Then, we check if `cpuTotal` exists for the current node. If it does not exist we add it to the `nodes` array with the current vCPU value. If it does exist we add the vCPU utilization to the existing value. Also, we store each vCPU utilization in a separate array within the current node.

6. **Calculate Bare Metal CPU utilization**
   To calculate the bare metal CPU utilization we use Equation 10. This equation gives us the total CPU utilization of the bare metal.

7. **Loop through nodes for memory usage metrics**
   We loop through the result of the node memory usage query. We add the memory usage of each node to the `nodesMemoryUsageBytes` variable. Also, we add the memory usage of the node to the `nodes` array with the current node value.

8. **Loop through nodes for memory total metrics**
   This step is almost the same as the previous one. We loop through the result of the node memory total query. We add the memory total of each node to the `nodesMemoryTotalBytes` variable. Also, we add the memory total of the node to the `nodes` array with the current node value.

9. **Calculate physical memory usage**
   Up till this point, we have the total memory usage of all nodes. We also have the total memory of all nodes. We can calculate the memory usage of the bare metal server by using Equation 16. This calculation uses the total memory usage of all known nodes and the total memory usage of all unknown nodes. This equation gives us the total memory usage of the bare metal machine.

10. **Set timestamps**
    We set timestamps so that we can use them in the coming steps. We use the timestamp of the last query and call it `nodeMetricTimeStamp`. Also, we set a constant timestamp of 15 seconds before the previous one. We call this `nodeMetricTimeStamp15SecondsAgo`. After these two timestamp values are initialized we calculate the difference between them. We call this `upTimeInHours`. This is the difference between the two timestamps in hours. For now, this value will always be the same because we loop every 15 seconds.

11. **Retrieve latest carbon intensity**
    In this step, we retrieve the latest carbon intensity from Electricity Maps. It is possible to get the carbon intensity for a specific country. In this case, we use the Netherlands. The carbon intensity is measured in `gCO2eq/kWh`. This is the average carbon intensity of the Netherlands for the current hour.

    This request is not called every time we loop. If the datetime of the latest response is older than 60 minutes, we need to get a new response. This is because Electricity Maps only has the carbon intensity of the latest hour. Then the response is stored in a variable called `latestElectricityMapsResponse` outside of the loop. However, for a backup value, we use the average carbon intensity in the Netherlands for the year 2023[54]. The latest carbon intensity is pushed to Prometheus.

12. **Estimate kWh and CO2e for Bare Metal**
    This is where we use the Cloud Carbon Footprint package. We use the OnPremise package to estimate the energy usage and carbon emissions of the bare metal infrastructure. The package uses the main

[54] `https://www.nowtricity.com/country/netherlands/`

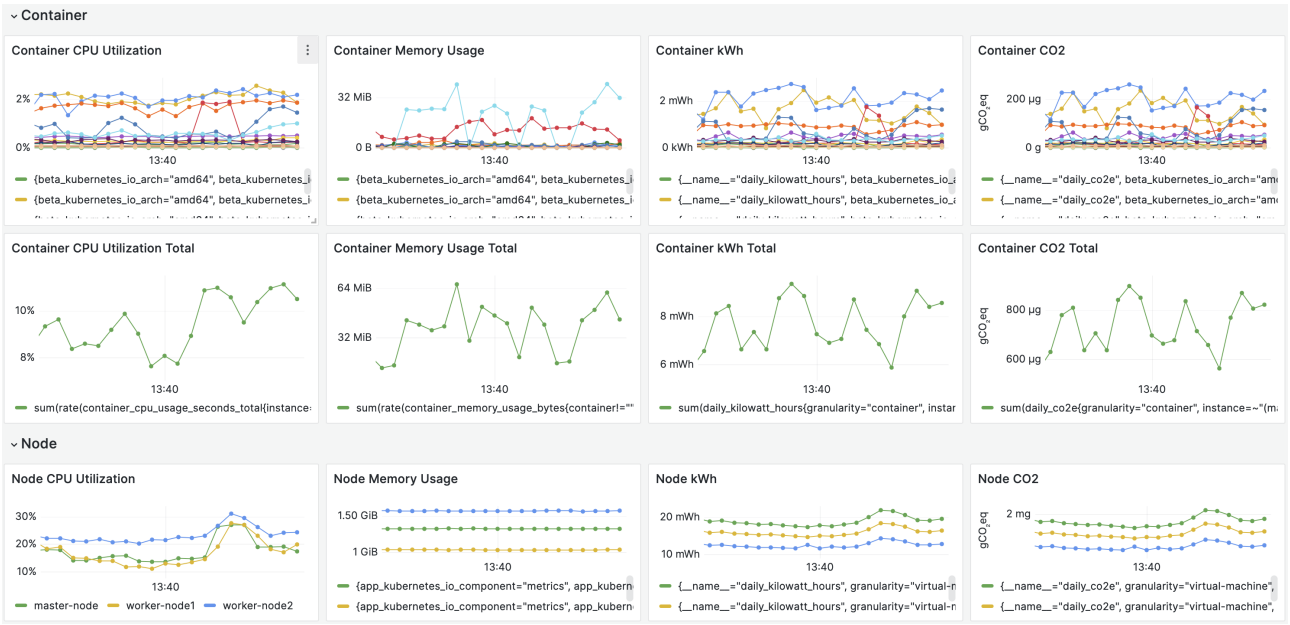


Figure 2: Grafana Dashboard — Container & Virtual Machine

methodology of the tool. This package needs the input parameters outlined in Table 8. We modified this package so that it accepts the latest carbon intensity as an input parameter. Hence, we can use the latest carbon intensity retrieved in the previous step. The package did not support dynamic carbon intensity. Therefore, we added this feature to the package.

For the memory input field, we use the MU_{BM} `/ 1000000000` to get the memory in gigabytes. This value is calculated in Equation 16. For the CPU input field, we use the U_{BM} from Equation 10.

Since some features are not yet supported in the OnPremise package. For example, it did not recognize a processor from the 3RD GEN AMD processor family. Also, it does not support using the Electricity Maps API for OnPremise. Therefore, we copied only the necessary files from the repository and used them in our project. This made the project more maintainable and easier to use.

Then we call the `getOnPremiseDataFromInputData` function. This function returns the same array as the input including kWh and CO2e. This function splits immediately into a `ComputeEstimator` and `MemoryEstimator`.

(a) `ComputeEstimator`
This class is used to estimate the energy usage and carbon emissions of the compute usage. Or in other words, CPU usage. First, it calculates the energy usage in kilowatt-hours. This energy formula can be found in Equations 23 and 17. Second, it calculates the carbon emissions in mtCO2e in Equation 19.

(b) `MemoryEstimator`
This part also starts with estimating the kilowatt hours. This is done by multiplying the gigabyte hours with multiple factors. The formula can be found in Equation 21. Then it calculates the carbon emissions in mtCO2e. This is done by multiplying the kilowatt hours with the carbon intensity. The formula can be found in Equation 22.

(c) `estimateCO2`
This function takes the estimated kilowatt hours and carbon intensity. The calculation is done by multiplying the kilowatt hours with the carbon intensity as seen in Equation 24.

After all of these calculations, it returns the kWh and CO2e for the given input parameters. This data is

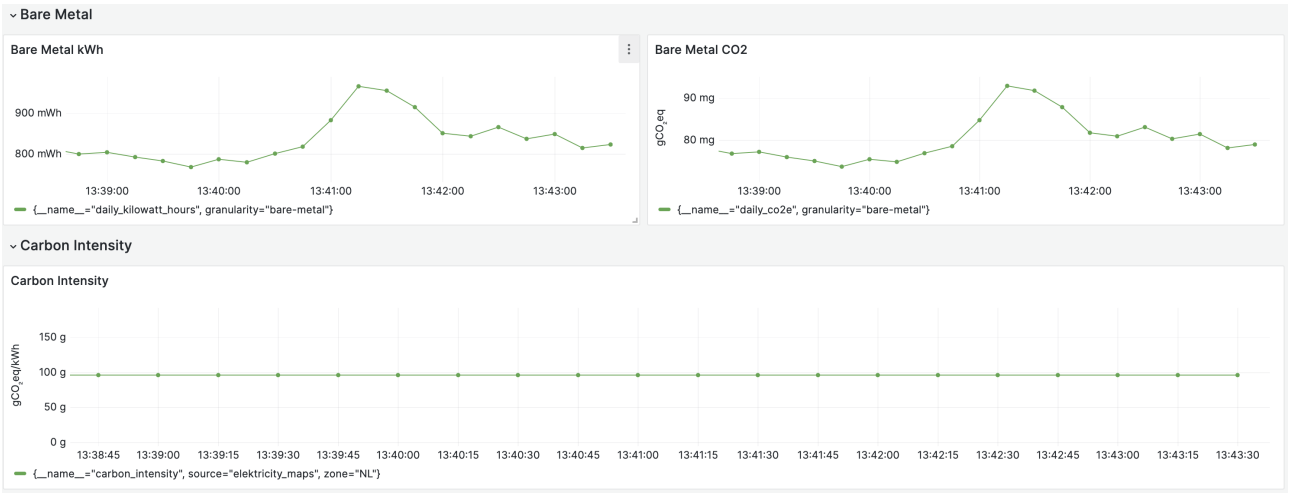


Figure 3: Grafana Dashboard — Bare Metal & Carbon Intensity

then pushed to Prometheus with the bare metal level abstraction label.

13. **Calculate Share for each Node**
    Now that we possess the total energy usage and carbon emissions of the bare metal we can calculate the share for each node. We loop through the *nodes* array that has all the node metrics at this point. A percentage is calculated for each node so that this percentage can be used to divide by the total energy usage and carbon emissions of the bare metal. This is shown in Equation 27. It takes the maximum of either the CPU or memory share. This is done by dividing the node CPU utilization by the total sum of all vCPU utilization values as shown in Equation 25. The same is done for the memory usage. This is done by dividing the node memory usage by the bare metal memory usage as shown in Equation 26. All results are then pushed to Prometheus with the node-level abstraction label.

14. **Query Container Metrics**
    In this step, we focus on the container metrics. We use the following queries to get the container metrics. The filter on the container name is used to prevent containers that are running in the background.

    - `rate(container_cpu_usage_seconds_totalcontainer!=""[1m])`
    - `rate(container_memory_usage_bytescontainer!=""[1m])`

15. **Loop through each container**
    The outcomes obtained from the queries undergo a looping process. We iterate through the vCPU results and locate the relevant memory result using the container ID. Utilizing these values, we compute the container's share of the node. This is achieved by dividing the container's CPU utilization by the node's CPU utilization. As demonstrated in Equations 32, 30, and 31 this conversion mirrors the transition from bare metal to the node. The results are then pushed to Prometheus with the container-level abstraction label.

16. **Start over**
    At this stage, all the data is transferred to Prometheus. We pause for 15 seconds before restarting the process.

4.3.2 Dashboard Implementation

The dashboard is implemented using Grafana. It is used to visualize the data that is stored in Prometheus. The dashboard is divided into four rows. The first row contains eight graphs about the container level. Four of them

display the CPU utilization, memory usage, energy consumption, and carbon emissions of each container. The other four display the same metric but for the total of all containers. The second row displays the same metrics for each node. The first and second rows can be seen in Figure 2. The third row displays only the total energy usage and carbon emissions of the bare metal. The fourth row is used to display the carbon intensity of the Netherlands. The third and fourth rows can be seen in Figure 3. All of these graphs are line charts that display the metric over time. The dashboard is refreshed every 15 seconds. This is the same interval as the loop in the model implementation. This means that the dashboard is updated every time the model is updated.

5 Validation

This section focuses on the validation of the model and its implementation given in Section 4. This validation is split into two parts. First, we will do a baseline measurement in Section 5.1. This measurement is taken at different times when there is no load on the cluster. This is done to see if the model and implementation are working correctly when there is no load on the cluster. Second, we will do a load measurement in Section 5.2. This is done to study how the model and implementation behave when there is load added to the cluster. Finally, we will summarize the results in Section 5.3.

Given that Electricity Maps provides carbon intensity measurements in grams of CO2 equivalent per kilowatt-hour (`gCO2eq/kWh`), it is evident that the intensity is already expressed in carbon dioxide equivalent format. Consequently, in the subsequent sections, we will employ `CO2eq` to denote the carbon footprint.

5.1 Baseline Measurement

This section will validate the results when there is no load on the cluster. This validation is done in iterations so that we can improve both the model and its implementation. It can be seen as a series of improvements, with each iteration representing a refinement over the previous one. The goal is to ensure consistency and accuracy by doing this iterative process of validation computations.

Important to note is that some modifications are already made to the model and implementation after these iterations. These changes are made based on the results of the iterations. The changes are explained in the respective iterations.

5.1.1 Iteration One

For this initial iteration, we sampled the results on `2024-02-07` within a time frame from `15:52:00` to `15:52:15`. Since data is pushed every 15 seconds, we have one value for each metric for each abstraction. The results are shown in Table 10.

Now that we have the results, we can start with the validation checks. If we take the `mgCO2eq` of all the three known nodes and add them up, we get a total of `2.76 mgCO2eq`. These nodes have `7 vCPUs` in total. So when we divide this total `mgCO2eq` by the number of `vCPUs`, we get `0.394 mgCO2eq/vCPU`.

We now possess the amount of `mgCO2eq` per `vCPU`. Subsequently, we can use this value to calculate the total `mgCO2eq` for the bare metal server. The bare metal server has `256 vCPUs`. When we multiply the `mgCO2eq/vCPU` of the known nodes by the total `vCPUs`, we get `100.9376 mgCO2eq`. This result is not equal to the total BM server `mgCO2eq` estimated by the CCF model, which is `136 mgCO2eq`. This means we have an error rate of `-25.781%`.

In addition, we can check if the total of the `vCPUs` of the nodes is equal to the total `vCPUs` of the server. When we divide the total `mgCO2eq` by the `mgCO2eq/vCPU`, we get `345 vCPUs`. This result is not equal to the total `vCPUs` of the server. Moreover, the calculated `vCPUs` has `89` more vCPUs than the total `vCPUs` of the server. The error rate for this calculation is `34.766%`.

Given the discrepancy between these findings, further investigation is needed. The `mgCO2eq` calculations do not align as expected, leading us to move to the next iteration hoping to improve the results.

	2024-02-07 (15:52:00 - 15:52:15)			
	CPU (%)	MEM (GiB)	mWh	mgCO2eq
server	**48.8**	**245.84**	**443.00**	**136**
master	**12.6**	**1.71**	**3.30**	**1.015**
calico-kube-controllers	0.13	0	0.0309	0.009
calico-node	1.82	0.00193987	0.518	0.159
coredns	0.369	0	0.082	0.0255

dashboard-metrics-scraper	0.00967	0	0.003	0.000956
kubernetes-dashboard	0.0204	0	0.002	0.002
metrics-server	0.211	0.00000232	0.057	0.017
node-exporter	0.0149	0	0.00521	0.0016
worker1	**21.6**	**1.7**	**3.29**	**1.012**
calico-node	1.71	0.00184	0.328	0.101
carbon-sight	0.207	0.00617188	0.023	0.00715
frontend-proxy	0.314	0	0.0436	0.013
node-exporter	0.0206	0	0.004	0.001
worker2	**10.5**	**1.23**	**2.38**	**0.733**
calico-node	1.64	0.00348	0.389	0.12
grafana	0.878	0.0000028	0.199	0.061
node-exporter	0.0109	0	0.00303	0.000933
prometheus-server	0.719	0.00787109	0.138	0.042

Table 10: Iteration One — Baseline Measurement — 2024-02-07 (15:52:00 - 15:52:15)

5.1.2 Iteration Two

The earlier iteration did not show confidence in the results. After some consideration, we realized that our assumption of the average vCPU utilization might be too optimistic. This encouraged us to explore alternative methods for calculating bare metal CPU utilization. Since we know the total number of known vCPUs and their respective CPU utilization percentages, it is possible to calculate the median value of the known vCPUs utilization. We added this approach to the model as an option, so that we can compare the results of the two methods. Therefore, the model and implementation sections were updated accordingly. Previously, our formula assumed a fixed CPU utilization percentage of 50% for all unknown vCPUs. In contrast, the new formula derives the CPU utilization percentage for unknown vCPUs from the median value of the known vCPUs.

After the formula was updated, we took another sample of the results on `2024-02-13` within a time frame from `11:11:00` to `11:11:15`. The sum of `mgCO2eq` for the three known nodes is `2.5 mgCO2eq`. This value comes close to the sum in the previous iteration. Again, dividing this value by the `7 vCPUs` utilized by the nodes, we get `0.357 mgCO2eq/vCPU`. This is slightly beneath the previous iteration's value of `0.394 mgCO2eq/vCPU`, but the reason for that can be the different time frames.

The total BM server `mgCO2eq` estimated by the CCF model is `86.1 mgCO2eq`. When we multiply the `mgCO2eq/vCPU` value by the same fixed number of vCPUs, we get `91.4 mgCO2eq` as can be seen in Equation 35. This result is considerably closer to the expected result than the previous iteration. The error rate between the `mgCO2eq` of the known nodes and the total BM server provided by the CCF model is `6.156%`. This is a significant improvement from the previous iteration.

In addition, when we look at Equation 36, we can see that the vCPUs based on the `mgCO2eq/vCPU` is `241 vCPUs`. This time the total calculated vCPUs is fewer than the actual number of vCPUs, which is `256`. The error rate is `-5.86%`.

This iteration shows some improvements, but we are not convinced yet. Also, since `kWh` will be more stable than `mgCO2eq`, we will assess both `mgCO2eq` and `kWh` in the next iteration.

$$0.357 \text{ mgCO2eq/vCPU} \times 256 \text{ vCPUs} = 91.4 \text{ mgCO2eq} \tag{35}$$

$$\frac{86.1 \text{ mgCO2eq}}{0.357 \text{ mgCO2eq/vCPU}} = 241 \text{ vCPUs} \tag{36}$$

5.1.3 Iteration Three

The prior iteration indicated advancements in the desired direction. However, we wanted to replicate the strategy employed for CPU utilization in evaluating memory usage. While utilizing the median for memory usage seemed impractical, we opted to experiment with the average of the known memory usage. Previously, we used half of the memory that was left for the unknown total memory. This was a very rough estimate, thus using the average of the known memory usage for the unknown memory usage seemed like a more accurate approach. This change is reflected in the model design and implementation by giving the option to use either the average or half of the memory for unknown memory usage.

Because of this change, we took another sample of the results. We sampled the results on `2024-02-13` within a time frame from `14:11:45` to `14:12:00`.

We will do validation computations for `mWh` first. The summed-up known nodes have a total of `13.73 mWh`. Dividing this by their `7 vCPUs`, we get `1.961 mWh/vCPU`, see Equation 37. Proceeding to calculate the total `mWh` for the bare metal server based on the `mWh/vCPU`, we get `502.016 mWh` as can be seen in Equation 38. This is higher than the total `mWh` of the bare metal server estimated by the CCF model, which is `435 mWh`. This gives an error rate of `15.4%`. In addition, when we look at Equation 39, we see that the calculated number of vCPUs based on the `mWh/vCPU` is `221.7 vCPUs`. This is lower than the actual number of vCPUs, which is `256`. The error rate is `-13.4%`. The previous iteration did not consider the `mWh`, so for this iteration, we can not compare these results with the previous iteration.

$$\frac{13.73 \text{ mWh}}{7 \text{ vCPUs}} = 1.961 \text{ mWh/vCPU} \tag{37}$$

$$1.961 \text{ mWh/vCPU} \times 256 \text{ vCPUs} = 502.016 \text{ mWh} \tag{38}$$

$$\frac{435 \text{ mWh}}{1.961 \text{ mWh/vCPU}} = 221.7 \text{ vCPUs} \tag{39}$$

When looking at `mgCO2eq` again, we have a total of `1.701 mgCO2eq` for the known nodes. In Equation 40, we divide this by the `7 vCPUs`. This gives us `0.243 mgCO2eq/vCPU`.

The total `mgCO2eq` for the bare metal server provided directly by the CCF model is `54 mgCO2eq`. When we multiply the `mgCO2eq/vCPU` by the total `vCPUs` in Equation 41, we get `62.208 mgCO2eq`. This number is higher than the `54 mgCO2eq` of the bare metal server. This gives an error rate of `15.2%`. While the previous iteration showed a trend in the right direction, the error rate of this iteration is higher again than the previous iteration. Looking at Equation 42, we see that the calculated number of vCPUs based on the `mgCO2eq/vCPU` is `222 vCPUs`. This is lower than the actual number of vCPUs, which is `256`. The error rate is `-13.281%`. Comparing this error rate with the previous iteration, we see that this error rate is higher.

$$\frac{1.701 \text{ mgCO2eq}}{7 \text{ vCPUs}} = 0.243 \text{ mgCO2eq/vCPU} \tag{40}$$

$$0.243 \text{ mgCO2eq/vCPU} \times 256 \text{ vCPUs} = 62.208 \text{ mgCO2eq} \tag{41}$$

$$\frac{54 \text{ mgCO2eq}}{0.243 \text{ mgCO2eq/vCPU}} = 222 \text{ vCPUs} \tag{42}$$

5.1.4 Iteration Four

Given that the formulas were updated to use the median for CPU and the average for memory, we decided to compare the results of the two different methods. Table 11 shows the results of the iteration. Since mgCO2eq is not practical for comparison due to the fluctuating carbon intensity, we focus only on energy from this point. While the CPU utilization differs greatly, the total `mWh` is approximately the same. While the half method has almost `6.6` times as much CPU utilization, the total `mWh` is only `9 mWh` higher. This shows that the different methods do not have a big difference or that there is a mistake in the implementation.

Calc.	vCPUs	Level	CPU (%)	MEM (GB)	mWh	mWh/ vCPU	vCPUs/ server	mWh/ server
Median	256	server	7.4	96.25	434	1.695	256	434
	7	VMs	-	5.449	24.49	3.499	124.051	895.634
		containers	-	-	4.826	0.689	629.507	176.494
Half	256	server	48.8	245.84	443	1.731	256	443
	7	VMs	-	4.64	8.97	1.281	345.708	328.046
		containers	-	-	1.82574	0.261	1698.489	66.77

Table 11: Iteration Four — Compare methods of calculating the bare metal CPU utilization

5.1.5 Iteration Five

Upon reviewing the implementation again, we noticed an error in the calculation of average watts. Equation 17 was not correctly implemented. The code mistakenly divided the CPU utilization of the bare metal server by 100, although it was already represented as a fraction ranging from 0 to 1. Essentially, this resulted in the CPU utilization being divided by 100 twice. This likely explains why the total `mWh` remained approximately the same in the previous iteration because the percentage is too small to make a difference. This is fixed and we took another sample of the results.

On top of that, in this phase of the iteration, it came to our attention that we could use more accurate minimum and maximum watts for the implementation since we know the CPU and memory specifications of the server. Therefore, we took all records from the SPECpower_ssj 2008[55] benchmark that have the same CPU. These records are shown in Table 9. Previously, the CCF package took the average of all values within the same CPU family, but we will only use the ones that have the same number of vCPUs/threads. From this point, the new values are used in the implementation.

This iteration focuses on the use of different numbers of vCPUs and the use of different CPU utilization calculation methods. The reason for this is to see how the results behave in terms of consistency when having a different number of total vCPUs. If the model and implementation are consistent, the energy consumption and carbon emissions should decrease respectively when having fewer vCPUs. Because memory usage is only considered in the formulas when the usage is higher than the average memory total of the SPECpower_ssj 2008 records, we will use only the average method for memory usage.

See Table 14 in Appendix A for the results of the iteration. It becomes clear that the CPU utilization is now reflected in the energy consumption. We already know that the CPU utilization is higher when using the average method, but now with the error fixed, we can see that the total energy consumption is also higher. In addition, the error rates have increased as well. When looking at the first row in Table 14, we see that the server has `820.222 mWh` estimated directly from the CCF model. The calculated server total based on the `mWh/vCPU` of the nodes, we get `1690.696 mWh`. This is an error rate of `106.6%`. This is a significant increase from iteration three, which only had an error rate of `15.4%`.

[55] `https://www.spec.org/benchmarks.html#power`

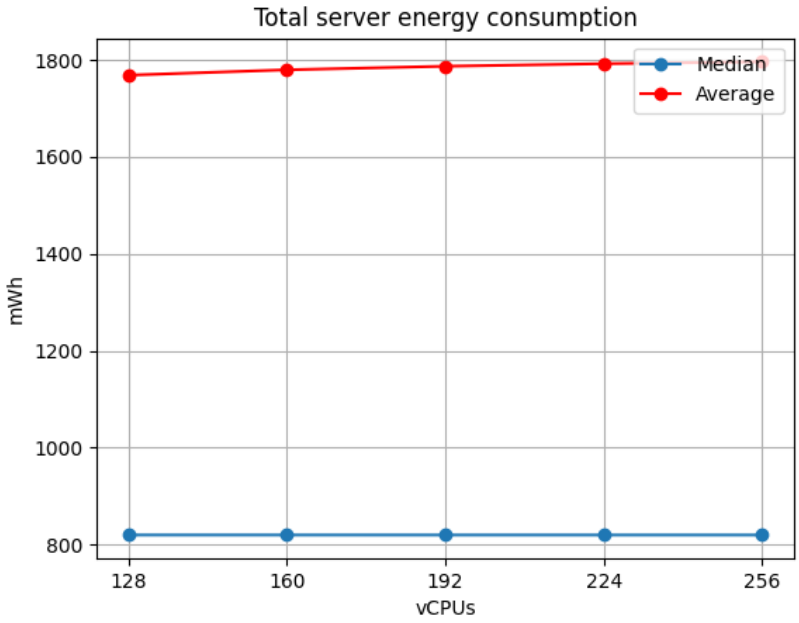


Figure 4: Iteration Five — Total energy consumption per server by different methods

In Figure 4, we see that the total energy consumption of the server is higher when the CPU utilization is calculated with the average of the known vCPUs. Also, the total energy consumption does not change when having different numbers of vCPUs for the median method. However, for the Average method, the total energy consumption is increasing very slightly when having more vCPUs, but this is minimal. The energy consumption does not change when having different numbers of vCPUs because the model does not consider the total number of vCPUs when calculating the energy consumption. The model will still use the same amount of min and max watts for the calculation, regardless of the number of vCPUs.

Figure 5, shows that both calculation methods are decreasing in the same way. Both have a lower `mWh/vCPU` when having more vCPUs. As mentioned in the previous paragraph, this is because the same amount of min and max watts is used for the calculation.

When reviewing Table 14 in Appendix A, particularly focusing on the last four columns, there appears to be a discrepancy in the results. Looking at the first row, which represents three different abstraction levels, is an example to illustrate this issue. This row indicates `256 vCPUs`. For the BM CPU utilization of `7.4%`, the energy consumption is recorded as `820.222 mWh`, resulting in `3.204 mWh/vCPU`. However, when we sum up the energy consumption for all known vCPUs, we obtain `46.230 mWh`, translating to `6.604 mWh/vCPU`. This is almost twice the initial `mWh/vCPU` for the BM.

The inconsistency may stem from the use of the median, which could inflate the total `mWh` at the VM level if one vCPU has disproportionately high utilization. Alternatively, we can explore the results of the average method, which assumes uniform vCPU utilizations. When looking at the first row that has `CPU_MEDIAN` set to False. We know that all the vCPU utilization values are lower than `50%`, so the average method should give us a lower `mWh/vCPU` at the VM level compared to the BM level. However, the data shows that the `mWh/vCPU` at the VM level is nearly twice that of the BM level. Once again, this inconsistency raises concerns about the reliability of the findings.

5.1.6 Iteration Six

The purpose of this iteration is to determine the energy consumption at both minimal and maximal CPU utilization levels. This check ensures that the results are within the boundaries of the minimum and maximum watts

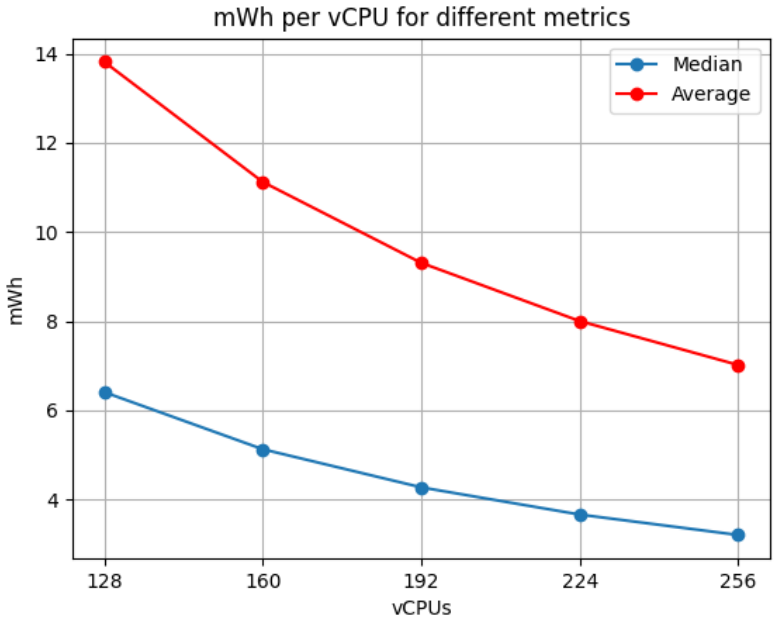


Figure 5: Iteration Five — Energy consumption per vCPU by different methods

of the server. Additionally, we aim to compare the outcomes obtained through the two methods of calculating CPU utilization.

The results of this iteration are presented in Table 12. It is evident from the table that both methods yield identical energy values (`mWh`) for corresponding CPU utilization levels. Specifically, the maximum energy consumption is recorded as `3000.7 mWh`, while the minimum stands at `646.5 mWh`. Furthermore, in Equation 43, we verify the energy consumption at 50% CPU utilization. We find that the result is `1823.6 mWh`, which is consistent with the values obtained from the table. This iteration did not result in any extra improvements.

$$\frac{(3000.7 - 646.5)}{2} + 646.5 = 1823.6 \tag{43}$$

CPU (%)	CPU_MEDIAN	mWh
100	False	3000.7
50	False	1823.6
0	False	646.5
100	True	3000.7
50	True	1823.6
0	True	646.5

Table 12: Iteration Six — Energy consumption bare metal server at different CPU utilizations

5.2 Load Measurement

In this section, we will study the behavior of the model and implementation by adding a load to the cluster. We do this to see if the energy consumption is increasing when the load is increasing. We will use different types of loads to see if the energy consumption is increasing in the same way. This should ensure that the model and implementation show consistent behavior aligning with the load that is added.

Timestamp	Users	RPS	mWh CON	mWh/RPS
22:23:29	1000	372.8	10.2	0.02736051502
22:18:19	694	260.1	7.07	0.02718185313
22:13:54	429	158.5	7.09	0.0447318612
22:08:44	119	41.5	1.96	0.04722891566
22:07:34	50	15.6	0.885	0.05673076923
22:06:59	15	2.9	0.051	0.0175862069
22:06:59	10	1	0.051	0.051

Table 13: Load Measurement — Locust — Energy consumption per request

5.2.1 Fibonacci Sequence

The first load we will add is a single container that calculates a Fibonacci sequence. This is a Python script that calculates the numbers and prints them to the console. This is a CPU-intensive task. Besides adding load, this test will also show that the implementation can detect a new container and start monitoring it.

As we can see in Figure 6 that is in the Appendix B, all the energy consumption is increasing. In the first Figure 6a, it is clear that the energy goes up when the container is deployed. The line fluctuates a bit around `800 mWh`, after the container is deployed, the line increases to `950 mWh` and reaches later the max of `1050`. Looking at Figure 6b, we can see that the energy is increasing from around `15 mWh` to `30 mWh`. This value is doubled. For the energy of the container, it is interesting to see that the energy is increasing later than the other two. At its highest point, it has an average of `20 mWh`.

Besides the energy consumption, the CPU utilization of the virtual machine is going over `100%`. However, it fluctuates between `100%` and `110%`. This needs further investigation because in all probability the percentage of the virtual machine is not normalized. For the container CPU utilization itself, it is going up to `100%`, but not beyond that. Lastly, when looking at the memory in Figure 6f, we can see that the memory is not increasing. This is because the memory is not used in this task.

Although this test cannot tell us whether the calculations are consistent and correct, it does show that the energy consumption is increasing when the load is increasing. On top of that, the monitor detects the new container and starts monitoring it.

5.2.2 Locust Load Test

This section will be responsible for showing the alignment between energy consumption and the incoming load to a Kubernetes cluster in the form of requests per second. We will use Locust[56] to generate the requests and send them to a `cartservice`. This `cartservice` is running in a container and is part of OpenTelemetry Demo[57] microservices application. The `cartservice` is a simple shopping cart service and is completely standalone. We can access the container through API requests. Both, GET and POST requests are sent.

With the setup in place, we will send requests to the `cartservice` and measure the energy consumption of the container, virtual machine, and bare metal. The Locust load test started at 10:06:49, hit the configured 1000 users at 10:23:29, and ended at 10:27:42. The energy consumption was measured during the entire test. The results are shown in Figure 7, where the `cartservice` container and its related virtual machine are shown in Figure 7a, and the bare metal is shown in Figure 7b.

It becomes clear that all the energy lines go up with the increasing number of requests respectively. The energy consumption of the container and virtual machine are close to each other, but the container energy consumption is always lower than the virtual machine where it is running. For each abstraction, it is clear that

[56] `https://locust.io`

[57] `https://github.com/open-telemetry/opentelemetry-demo`

the energy consumption drops immediately after the requests are stopped. This is because the CPU is not being used anymore, and the energy consumption is reduced.

We took some random points in time to calculate the energy consumption per request per second. The results are shown in Table 13. The results show a trend in `mWh/RPS`. The energy consumption per request per second is decreasing when the number of requests per second is increasing. However, there are also some outliers. The `mWh/RPS` at `22:06:59` is lower than the one at `22:07:34`, but the `RPS` is higher.

5.3 Summary

The validation process provided insights into the model and its implementation. Through iterative evaluations, we aimed to refine the reliability and consistency. Each iteration involved validation computations and adjusting methodologies.

In the initial iterations, discrepancies between the total `mWh` and `mgCO2eq` of the bare metal provided directly by the CCF model and the calculated values based on the known vCPUs were observed. The error rates were significantly high, indicating the need for adjustments. We introduced new methods for calculating CPU and memory utilization, which did not yield the desired results. The error rates remained high, suggesting that the model's predictions were inconsistent and unreliable.

Further refinements were made in later iterations, including corrections to implementation errors and the implementation of more precise benchmarks for power consumption. Despite these efforts, the results remained inconsistent, with error rates fluctuating between iterations. Overall, the validation process provided valuable insights into the limitations of the model and its implementation. While adjustments were made to improve the model and implementation, further improvement is needed.

6 Conclusion

This section will conclude the thesis by summarizing the results and answering the research questions. After that, a section about the lessons that we learned will be presented, followed by a section on future work.

6.1 Summary

The project initiated by Digital Lab aimed to monitor the carbon emissions of containers in a Kubernetes environment. Despite efforts to explore methods and tools for calculation and validation, the constraints and requirements mentioned in Section 3.3 persisted. With most of the limitations, there were options available, except for the constraint of not having access to the hypervisor. This was the biggest constraint and the main reason for the challenges we faced. This resulted in our inability to monitor the complete physical server and its usage, but only the virtual machines that we have access to. Consider it as attempting to estimate the energy consumption of an entire hotel only by examining the energy usage within the rooms where access is permitted. This is a major limitation, as we can not see the entire picture and therefore can not make accurate estimates. When we had access to the hypervisor, we would have a ground truth to compare with, and we could validate the results more accurately.

This led to the adoption of an exploratory approach, relying solely on known virtual machine metrics to estimate energy consumption and carbon emissions at the bare metal server level. These estimates were then used to allocate carbon emissions among the virtual machines and containers.

However, the validation of the results did not demonstrate internal consistency. Section 5.1.5 demonstrated that the results from different levels of abstraction did not add up correctly. For each level of abstraction, we converted the total energy to `mWh/vCPU`. This value at the VM level was higher than at the bare metal server level, and the value at the container level was much lower than both the VM and BM level. This inconsistency was an indication that the results were not reliable.

The virtual machine metrics on which the approach relies do not provide sufficient information to estimate energy consumption at the bare metal level. This limitation arises because a virtual machine operates within its isolated environment, without considering external factors. Consequently, monitoring carbon emissions at the container level in a virtualized environment without physical hardware or hypervisor access remains inconsistent. Access to the hypervisor is crucial to observe the actual utilization of physical hardware and obtain reliable and accurate results.

6.2 Answers to Research Questions

1. *How can the carbon footprint of individual containers be calculated in virtualized environments?*
   The literature review has revealed various methods and tools for calculating the carbon footprint in virtualized environments. However, most of these methods and tools necessitate access to physical hardware, hypervisor, or administrative rights on OpenStack.

   In particular, when it comes to monitoring energy consumption at the container level in a virtualized environment there are several tools available. In Section 3.5.2, we discussed impressive tools such as Scaphandre, Kepler, Containergy, and PowerAPI. These tools come with built-in capabilities for analyzing energy consumption. Nonetheless, they do rely on having access to the hypervisor. In contrast, these do not need access to the physical hardware or administrative rights on OpenStack.

   Given the specific requirements and constraints of our use case listed in Section 3.3, the options were limited. To address this, we adopted an exploratory approach. This approach relies solely on virtual machine metrics to estimate energy consumption and carbon emissions at the bare metal server level. These estimates are then used to allocate carbon emissions among the virtual machines and containers.

   While the results from different levels of abstraction roughly correspond to the system's load, the validation, unfortunately, did not demonstrate internal consistency, definitely indicating a lack of accuracy. The virtual machine metrics on which the approach relies do not provide sufficient information to estimate

energy consumption at the bare metal level. This limitation arises because a virtual machine operates within its isolated environment, without considering external factors, it is like an island unto itself.

When we had access to the hypervisor, we would have a ground truth to compare with, from there we could see if the results were accurate. However, we would still have to deal with the challenge of converting energy consumption and carbon emissions from the bare metal server level to the virtual machine and container level. This is where most of the inconsistencies arose.

Consequently, monitoring carbon emissions at the container level in a virtualized environment without physical hardware or hypervisor access remains inconsistent. Access to the hypervisor is crucial to observe the actual utilization of physical hardware and obtain reliable and accurate results.

2. *How can the reliability and precision of carbon emission calculations be assessed and validated to ensure consistency and accuracy in results?*
   Section 5 focused on validating the reliability and precision of the carbon emission calculations that were introduced in Section 4. The first and most important step was to show consistency when there was no load on the system. We took data samples at different times from different levels of abstraction and compared the results. These comparisons started with sanity checks, such as ensuring that the results from the different levels of abstraction add up to the same value. This already showed inconsistencies, as for each level of abstraction, we converted the total energy to `mWh/vCPU`, `vCPUs/server`, and `mWh/server`. These values for VM and CON did not add up to the BM level. This indicated that either the the estimation of the energy consumption at BM level was wrong, or the distribution of the energy consumption from BM to VM and CON was wrong.

   The next step was to compare the results from the different levels of abstraction with the actual load on the system. This was done by running a load test on the system and comparing the results from the different levels of abstraction with the actual load. The results from the different levels of abstraction were consistent with the actual load on the system.

   In addition to the inconsistencies, we cannot confirm the accuracy of the results. There was no possibility to validate the results on accuracy since we did not have any ground truth to compare with.

3. *How to monitor and visually represent carbon emissions at various levels of abstraction and raise users' awareness?*
   Section 4.3.2 explains and demonstrates the implementation of the dashboard. The dashboard was structured to see the usage at different levels of abstraction. In one real-time dashboard, the user can see the carbon emissions at the bare metal server level, the virtual machine level, and the container level. The dashboard updates every 15 seconds, this is aligned with the implementation of the model that updates every 15 seconds.

   The dashboard was designed to be user-friendly and intuitive. Users can view carbon emissions in grams of CO2 emitted and energy consumption in kWh. Observing these amounts helps with understanding the impact of actions on the environment. It encourages consideration of ways to consume less energy and hence reduce the carbon footprint, benefiting the planet. For cloud computing, it assists in making informed decisions regarding application design and deployment.

6.3 Lessons Learned

Despite our best efforts, we faced several limitations throughout this project. We tried various formula alternatives to translate from known vCPUs to the physical CPU, but did not achieve the desired success. We could have explored other energy models, such as standalone ones found in our literature review, but due to time constraints, this was not an option. Furthermore, it came to our attention that the total vCPU utilization values of virtual machines might not undergo normalization, potentially impacting the reliability of our results. This is because all the vCPUs within a virtual machine are aggregated. The absence of normalization in the total vCPU utilization of virtual machines could significantly influence the entire procedure. Given that we rely on VM

CPU utilization to approximate energy consumption at the bare-metal server level and subsequently allocate it to the VM and container level, inaccuracies may arise as a consequence.

Unfortunately, due to time limitations, we could not investigate this further. However, we believe that even with alternative energy models, our challenge persists because we only have access to a fraction of the physical server.

Even if our results were consistent, we could not validate their accuracy due to the lack of ground truth for comparison, representing another major limitation. Moreover, there are still factors that we did not take into account. One example is that we did not include energy consumption related to fans, storage, and network. Although these resources are not major contributors to energy consumption, they still contribute to overall energy consumption. Previous studies, such as the work by Westin [22], also pointed out difficulties in achieving high accuracy in their proof of concept. The main reason for uncertainty stemmed from their simulation not being in a controlled environment. They also utilized a virtual machine environment, similar to our approach. This further emphasizes the complexity of the problem.

Our model is built on several assumptions, assuming that unknown vCPUs are either half utilized or share the same utilization as known vCPUs, leading to a cascade of assumptions. Despite these assumptions, our results can still be used as rough estimates. By using minimum and maximum watt values as input, our energy model ensures that results remain within the boundaries of physical server energy consumption. This can be valuable, particularly in scenarios where hypervisor and hardware access is restricted, like cloud hosting, providing a rough idea of the carbon footprint.

A key lesson learned is the necessity of hypervisor access to address this problem, as concluded in the previous section. Despite our exhaustive efforts, unfortunately, success was not achieved. Nevertheless, the insights gained from this project contribute to the ongoing discussion on energy-efficient computing, emphasizing the need for further research in this field. We hope that our findings will inspire future research to explore alternative methods.

6.4 Future Work

Several areas could benefit from further research. This section will delve into potential future enhancements for the project.

The first crucial step involves investigating the consistency of the results. It is essential to stabilize this aspect before delving into ensuring result accuracy. Going over the entire process and identifying potential sources of inconsistency would be a good starting point. This could involve revisiting the vCPU normalization of virtual machines, as previously noted in Section 6.3. This issue can be addressed by looking at how the various vCPU utilization values of virtual machines are aggregated, considering that a virtual machine may possess multiple vCPUs.

Since the number of vCPUs per CPU is arbitrary, it would be interesting to explore the possibility of using a variable number of vCPUs in the energy model to calculate the energy consumption of the bare metal server. With this variable number of vCPUs, we can explore using different minimum and maximum watt values for varying numbers of vCPUs. By examining the SPECpower_ssj2008 database there are records with different setups. The records can have the same CPU but different numbers of vCPUs. Aligning our number of vCPUs with the records that have the same number of vCPUs, we can utilize the average watt values of those records as input in the energy model.

At the end of this thesis, a new pre-trained model was introduced that can be used for Kepler. This model is also based on the SPECpower_ssj2008 benchmark and is available on the Kepler Model Database repository[58]. Unfortunately, we did not have time to test this model, but this shows potential to use Kepler without needing hypervisor access. Therefore, we recommend testing this model in the future.

As discussed in the related work section, Scaphandre has potential plans for employing an alternative approach without requiring hypervisor access. This potential approach may involve utilizing a look-up database

[58] https://github.com/sustainable-computing-io/kepler-model-db/tree/main/models/v0.7

with the energy consumption of different workloads, similar to the SPECpower_ssj2008 benchmark mentioned in this thesis. This is a development worth monitoring closely to determine its applicability in the future.

We demonstrated that Scaphandre only operates when paired with an Intel CPU, primarily due to its reliance on the Intel RAPL tool. Interestingly, during our research, we encountered an existing Intel RAPL driver tailored for AMD CPUs[59]. The availability of this driver raises the possibility of resolving compatibility concerns with AMD CPUs. Nonetheless, it is essential to note that Scaphandre is not designed to work on a virtual machine only. Despite this limitation, it is worth exploring whether Scaphandre can be used to monitor energy consumption on a virtual machine in combination with this driver.

Advancing further, incorporating energy consumption monitoring at the cluster level emerges as a logical progression in this study. Although it does not directly address the consistency issue, integrating this aspect would enhance the project. By doing so, we can effectively track separate clusters, as applications often utilize different clusters for various functionalities. Themis, for example, where one cluster handles code submissions while another has the pipelines to support this. This approach enables us to detect areas where optimizations are feasible. Additionally, when a single cluster operates on a physical server, it is impractical to attribute the entire server's energy consumption to the cluster due to inherent overhead utilization that may not directly correlate to the cluster's activities.

Further development of the dashboard is also a potential area for future work. Adding comparison graphs to the dashboard would be a good addition to the dashboard to compare the grams of CO2 emitted with something that the user can relate to. For example, compare the grams of CO2 emitted with the number of trees that would be needed to absorb the CO2. Or compare the grams of CO2 emitted with the number of kilometers driven by an electric car. This would make the results more understandable and relatable to the user. Eventually, integrating the dashboard with Themis to display the carbon footprint of individual code submissions to users could be explored.

[59] `https://github.com/amd/amd_energy`

Bibliography

[1] R. Radersma, "Green coding: Reduce your carbon footprint," *Ethical Software Engineering and Ethically Aligned Design*, pp. 19–20, 2022.

[2] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey, "Recalibrating global data center energy-use estimates," *Science*, vol. 367, no. 6481, pp. 984–986, 2020.

[3] R. Westerhof, "A model for the total carbon footprint of a cloud-based software as a service provider and its customers," 2022.

[4] G. Araújo, A. Sabino, L. Lima, V. Costa, C. Brito, P. Rego, I. Fé, and F. A. Silva, "Energy consumption in microservices architectures: A systematic literature review," 2023.

[5] W. Silva-de Souza, A. Iranfar, A. Bráulio, M. Zapater, S. Xavier-de Souza, K. Olcoz, and D. Atienza, "Containergy—a container-based energy and performance profiling tool for next generation workloads," *Energies*, vol. 13, no. 9, p. 2162, 2020.

[6] K. Haghshenas, B. Setz, and M. Aiello, "Co2emission aware scheduling for deep neural network training workloads," pp. 1542–1549, Institute of Electrical and Electronics Engineers Inc., 2022.

[7] Z. Meng, W.-B. Li, C. Chen, and C. Guan, "Carbon emission reduction effects of the digital economy: Mechanisms and evidence from 282 cities in china," *Land*, vol. 12, no. 4, p. 773, 2023.

[8] G. G. Protocol, "Greenhouse gas protocol," *Sector Toolsets for Iron and Steel-Guidance Document*, 2011.

[9] A. Dijkstra, "An improved model for the allocation of carbon emissions among the tenants of cloud services," 2023.

[10] A. James and D. Schien, "A low carbon kubernetes scheduler.," in *ICT4S*, 2019.

[11] T. C. Trust, "A guide to carbon footprinting for businesses," 2023.

[12] H. AlJahdali, A. Albatli, P. Garraghan, P. Townend, L. Lau, and J. Xu, "Multi-tenancy in cloud computing," in *2014 IEEE 8th international symposium on service oriented system engineering*, pp. 344–351, IEEE, 2014.

[13] A. Azeez, S. Perera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne, S. Weerawarana, and P. Fremantle, "Multi-tenant soa middleware for cloud computing," in *2010 IEEE 3rd international conference on cloud computing*, pp. 458–465, IEEE, 2010.

[14] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, pp. 732–794, 2016.

[15] W. Lin, W. Wu, H. Wang, J. Z. Wang, and C.-H. Hsu, "Experimental and quantitative analysis of server power model for cloud data centers," *Future Generation Computer Systems*, vol. 86, pp. 940–950, 2018.

[16] A. E. H. Bohra and V. Chaudhary, "Vmeter: Power modelling for virtualized clouds," in *2010 ieee international symposium on parallel & distributed processing, workshops and phd forum (ipdpsw)*, pp. 1–8, Ieee, 2010.

[17] M. Dullak, N.-B. Margotti, S. Zindl, M. Aiello, and B. Setz, "A review of modelling approaches for computer system components," 2022.

[18] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," in *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pp. 303–314, 2005.

[19] R. Lent, "Analysis of an energy proportional data center," *Ad Hoc Networks*, vol. 25, pp. 554–564, 2015.

[20] A.-C. Orgerie, L. Lefèvre, and I. Guérin-Lassous, "Energy-efficient bandwidth reservation for bulk data transfers in dedicated wired networks," *The Journal of Supercomputing*, vol. 62, pp. 1139–1166, 2012.

[21] A. Noureddine, R. Rouvoy, and L. Seinturier, "A review of energy measurement approaches," *ACM SIGOPS Operating Systems Review*, vol. 47, no. 3, pp. 42–49, 2013.

[22] J. Westin, "Evaluation of energy consumption in virtualization environments: proof of concept using containers," 2017.

[23] G. A. Brady, N. Kapur, J. L. Summers, and H. M. Thompson, "A case study and critical assessment in calculating power usage effectiveness for a data centre," *Energy Conversion and Management*, vol. 76, pp. 155–161, 2013.

[24] H. S. Matthews, C. T. Hendrickson, and C. L. Weber, "The importance of carbon footprint estimation boundaries," 2008.

[25] T. Wiedmann and J. Minx, "A definition of "carbon footprint" isa uk research report 07-01. isauk research & consulting," 2007.

[26] A. Radovanović, R. Koningstein, I. Schneider, B. Chen, A. Duarte, B. Roy, D. Xiao, M. Haridasan, P. Hung, N. Care, *et al.*, "Carbon-aware computing for datacenters," *IEEE Transactions on Power Systems*, vol. 38, no. 2, pp. 1270–1280, 2022.

[27] J. Flinn and M. Satyanarayanan, "Powerscope: A tool for profiling the energy usage of mobile applications," in *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*, pp. 2–10, IEEE, 1999.

[28] T. Do, S. Rawshdeh, and W. Shi, "ptop: A process-level power profiling tool," 2009.

[29] W. Lin, H. Wang, Y. Zhang, D. Qi, J. Z. Wang, and V. Chang, "A cloud server energy consumption measurement system for heterogeneous cloud environments," *Information Sciences*, vol. 468, pp. 47–62, 2018.

[30] J. W. Smith, A. Khajeh-Hosseini, J. S. Ward, and I. Sommerville, "Cloudmonitor: Profiling power usage," in *2012 IEEE Fifth International Conference on Cloud Computing*, pp. 947–948, IEEE, 2012.

[31] R. Rosen, "Resource management: Linux kernel namespaces and cgroups," *Haifux, May*, vol. 186, p. 70, 2013.

[32] L. Brown, "Acpi in linux," in *Linux Symposium*, vol. 51, p. 20, 2005.

[33] M. Xu and R. Buyya, "Managing renewable energy and carbon footprint in multi-cloud computing environments," *Journal of Parallel and Distributed Computing*, vol. 135, pp. 191–202, 2020.

[34] K. Kaur, S. Garg, G. Kaddoum, S. H. Ahmed, and M. Atiquzzaman, "Keids: Kubernetes-based energy and interference driven scheduler for industrial iot in edge-cloud ecosystem," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4228–4237, 2019.

[35] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, and W. Lintner, "United states data center energy usage report," 2016.

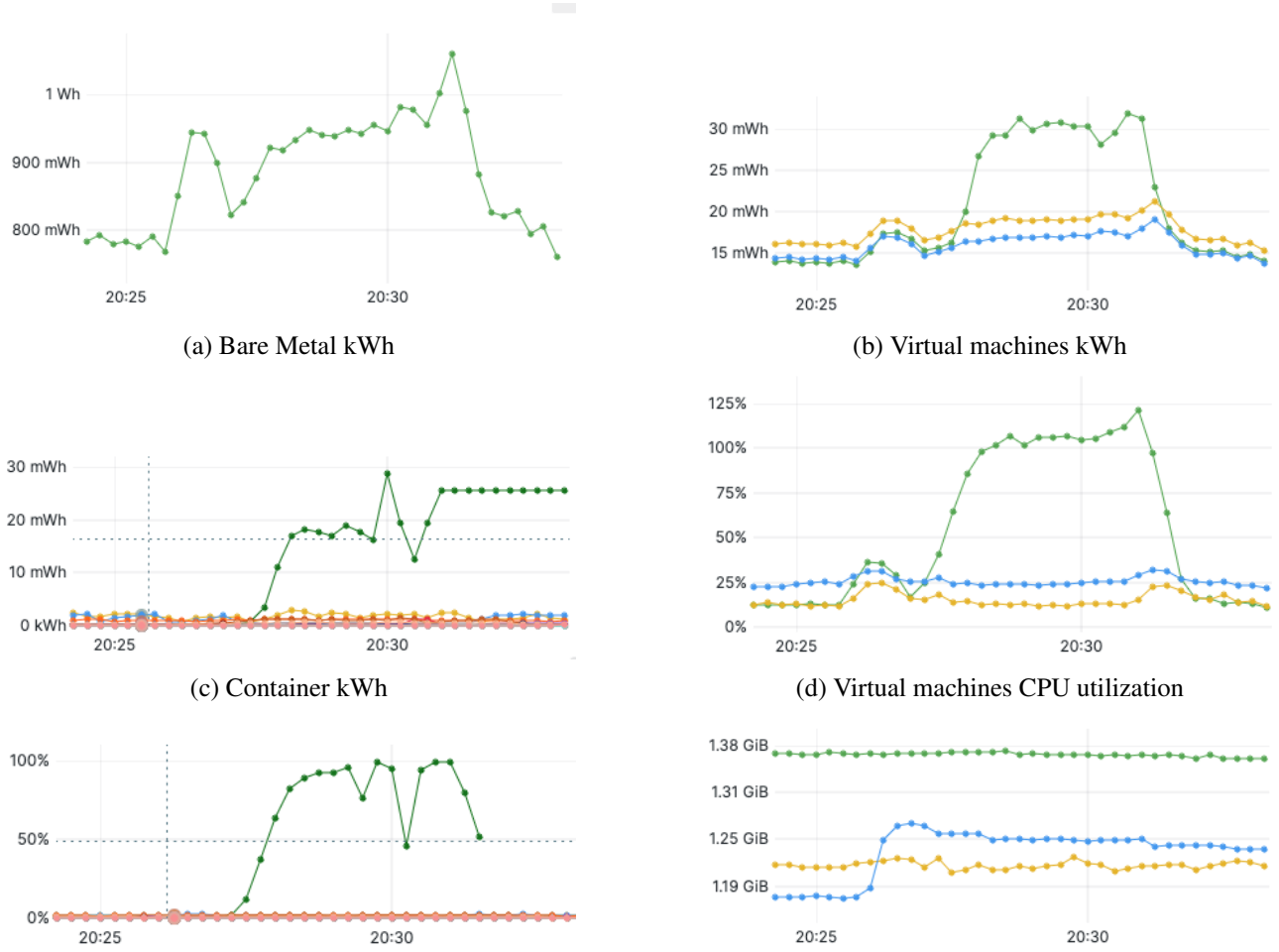
Appendices

A Energy consumption per server, VM and container with different number of vCPUs

CPU MEDIAN	MEM AVG	vCPUs	Level	CPU (%)	MEM (GB)	mWh	mWh/ vCPU	vCPUs/ server	mWh/ server
True	True	256	server	7.4	92.770	820.222	3.204	256	820.222
		7	vms			46.230	6.604	124	1690.696
		7	containers			6.732	0.962	853	246.191
True	True	224	server	7.4	92.770	820.229	3.662	224	820.229
		7	vms			46.230	6.604	124	1479.372
		7	containers			6.732	0.962	853	215.417
True	True	192	server	7.4	92.770	820.239	4.272	192	820.239
		7	vms			46.231	6.604	124	1268.048
		7	containers			7.666	1.095	749	210.260
True	True	160	server	7.4	92.770	820.253	5.127	160	820.253
		7	vms			46.232	6.605	124	1056.724
		7	containers			7.662	1.095	749	175.124
True	True	128	server	7.4	92.770	820.273	6.408	128	820.273
		7	vms			50.075	7.154	115	915.665
		7	containers			8.142	1.163	705	148.879
False	True	256	server	48.8	92.694	1795.956	7.015	256	1795.956
		7	vms			101.225	14.461	124	3701.944
		7	containers			19.662	2.809	639	719.062
False	True	224	server	48.8	92.694	1792.009	8.000	224	1792.009
		7	vms			101.003	14.429	124	3232.082
		7	containers			19.611	2.802	640	627.544
False	True	192	server	48.8	92.694	1786.747	9.306	192	1786.747
		7	vms			100.706	14.387	124	2762.221
		7	containers			19.546	2.792	640	536.114
False	True	160	server	48.8	92.694	1779.380	11.121	160	1779.380
		7	vms			100.291	14.327	124	2292.360
		7	containers			19.459	2.780	640	444.770
False	True	128	server	48.8	92.694	1768.329	13.815	128	1768.329
		7	vms			99.668	14.238	124	1822.498
		7	containers			19.331	2.762	640	353.482

Table 14: Iteration Five — Energy consumption per server, VM and container with different number of vCPUs

B Fibonacci Load Measurement



(a) Bare Metal kWh

(b) Virtual machines kWh

(c) Container kWh

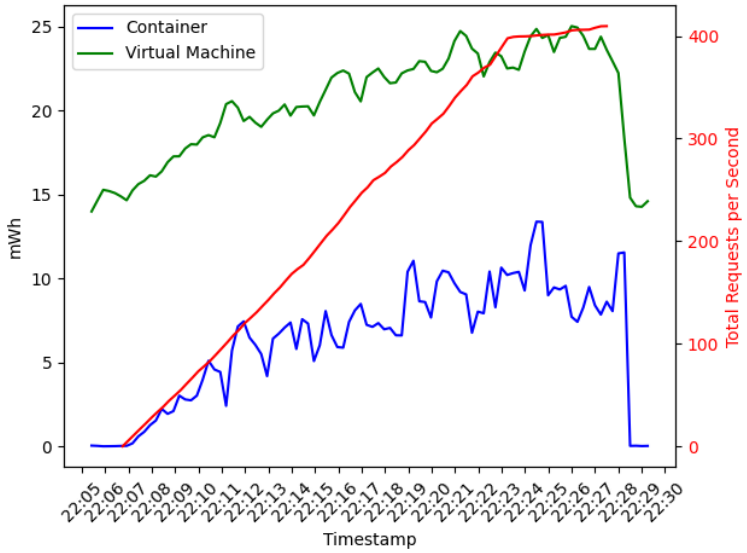
(d) Virtual machines CPU utilization

(e) Container CPU utilization

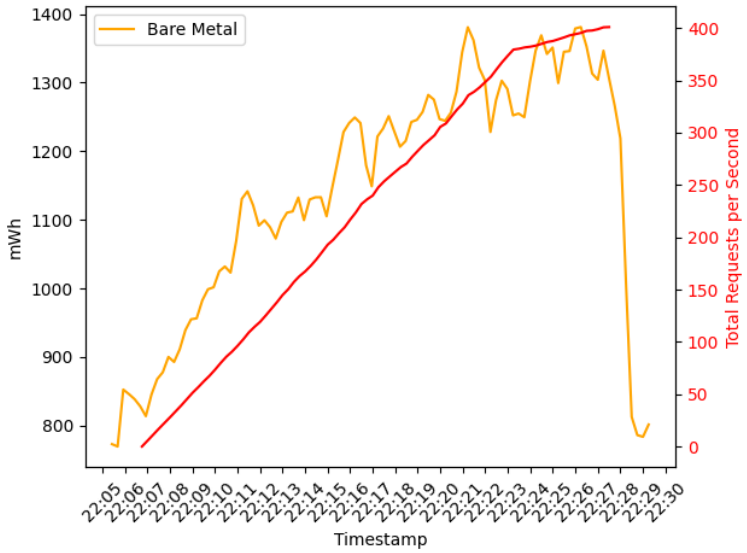
(f) Virtual machines memory utilization

Figure 6: Load Measurement — Fibonacci Sequence

C Locust Load Measurement



(a) Container & Virtual Machine



(b) Bare Metal

Figure 7: Load Measurement — Locust — Energy Consumption and Requests per Second