Aalto University
School of Science

Master’s Programme in Computer, Communication and Information Sciences

Towards Carbon-aware: Power Monitoring in Cloud Computing

Nhut My Cao

Master’s Thesis
2024

© 2024

This work is licensed under a Creative Commons "Attribution-NonCommercial-ShareAlike 4.0 International" license.

Aalto University
School of Science

Author Nhut My Cao

Title Towards Carbon-aware: Power Monitoring in Cloud Computing

Degree programme Computer, Communication and Information Sciences

Major Security and Cloud Computing

Supervisor Senior Lecturer Vesa Hirvisalo

Advisor M.Sc. Anton Debner

Date 29th December 2024 **Number of pages** 49+3 **Language** English

Abstract

The escalating adoption of cloud computing has not only bring significant values in terms of technology, but also sparked critical concerns regarding its environmental footprint. Together with the advantages cloud services provide, there exist issues from the energy consumption management and its impacts on the planet. The vast data centers required to support cloud services consume significant amounts of energy, much of which is derived from non-renewable sources, contributing to greenhouse effect. Additionally, the manufacturing, operation, and disposal of hardware result in substantial ecological impacts, including resource depletion and electronic waste. The thesis delves into to the energy efficiency and sustainability of cloud computing environments by analyzing data collected from monitoring tools deployed in customized cluster, aiming to examine the role of monitoring tools in managing computing systems.

The research centers on comprehending the functionality of monitoring tools and assessing their performance within this context. By scrutinizing key metrics such as CPU time, CPU cycles, and energy consumption patterns, the study provides insights into the factors that influence overall system performance and costs.

The findings underscore the potential of monitoring tools such as Kepler to optimize resource allocation and enhance energy efficiency. The study also identifies limitations in the current state of monitoring tools and emphasizes the necessity for further development to capture a more representative set of system metrics. The analysis highlights the significance of exploring innovative approaches to sustainable cloud computing, such as developing and committing to energy-efficient architectures, optimizing cooling systems, and implementing effective methods to achieve the sustainability.

Keywords Cloud computing, energy efficiency, monitoring tools, Kepler, sustainability, data centers

Preface

I want to thank my supervisor Vesa Hirvasalo and my instructor M.Sc. Anton Debner for their guidance and help.

I also want to thank my family and friends for keeping me sane and alive.

Otaniemi, 29th December 2024

Nhut M. Cao

Contents

Abstract 3

Preface 4

Contents 5

1 Introduction 7
- 1.1 Research Objectives . . . 7
- 1.2 Thesis Contribution . . . 8
- 1.3 Structure of the Thesis . . . 9

2 Background 10
- 2.1 Cloud computing and data centers . . . 10
- 2.2 Energy monitoring . . . 10
- 2.3 Carbon emission in cloud computing . . . 11
- 2.4 Preview of experiment . . . 12

3 State-of-the-Art in System Monitoring and Sustainable Data Centers 12
- 3.1 Tracing tools in OS . . . 13
  - 3.1.1 DTrace . . . 14
  - 3.1.2 SystemTap . . . 14
  - 3.1.3 LTTng . . . 15
  - 3.1.4 BPF and eBPF . . . 15
- 3.2 Hardware monitoring by performance counters . . . 16
- 3.3 Data processing for ML pipelines . . . 17
- 3.4 Cloud computing and the sustainability . . . 18
- 3.5 Data centers in Nordics . . . 19

4 Experimental tools 21
- 4.1 Kubernetes . . . 21
- 4.2 Kepler . . . 22
- 4.3 Prometheus . . . 23
- 4.4 Grafana . . . 23
- 4.5 Apache Kafka . . . 24

5 Experiment: Objective and Configuration 24
- 5.1 Machine configuration . . . 25
  - 5.1.1 Linux and eBPF . . . 25
  - 5.1.2 RAPL . . . 26
- 5.2 Software Configuration . . . 26
  - 5.2.1 Kubernetes . . . 27
  - 5.2.2 Kepler . . . 27
  - 5.2.3 Cluster configuration . . . 27
  - 5.2.4 Feeding workload . . . 29

5.2.5 Execution 30

6 Experiment Result 30
6.1 Overview of collected data 32
6.2 Data analysis 34

7 Discussion 38
7.1 Experimental Methodology 38
7.2 Advantages and disadvantages of Kepler 38
7.3 Connection with the carbon emission 40
7.4 The sustainability challenge in cloud computing 41

8 Conclusions 42
8.1 Summary of findings 42
8.2 Future work and improvements 43

References 44

A Appendix: Customed Workload Script 50

1 Introduction

The modern technological world has witnessed the unprecedented development of cloud computing, with its popularity and growth exceeding all expectations. The extensive benefits and helpful assistance of cloud technology are shown in the raising popularity of cloud computing keyword. With the presence of cloud computing, the way businesses and individuals interact with technology has changed considerably. By providing on-demand access to a wide variety of computing resources over the internet with affordable fees, cloud computing has significantly reduced the need for physical hardware and software installations. This paradigm shift has led to the escalation of data centers, hyper-scale facilities housing thousands of servers that power cloud services.

Cloud computing offers many benefits, including scalability, cost-effectiveness, and high-quality performance. Businesses can effortlessly scale their computing resources as their needs to meet fluctuating demands, avoiding the need for costly upfront investments in hardware. Additionally, cloud providers often offer pay-as-you-go pricing models, allowing organizations to optimize their spending based on actual usage. Furthermore, cloud-based services are typically reliable and performant, leveraging advanced technologies such as virtualization, load balancing, and spare infrastructure to handle unexpected corruptions and ensure seamless operations.

In order to support the demanding requirements of modern cloud services, data centers have undergone significant technological advancements. High-speed networking, advanced cooling systems, and energy-efficient hardware are essential components of these large-scale data centers. Data centers operate continuously 24/7, and are designed to handle unexpected surges in workload, requiring substantial energy consumption. Globally, data centers consume up to 200 terawatts per hour, with major providers like Amazon and Microsoft accounting for approximately 45% of this value [9]. The energy sources powering data centers vary widely depend on the nature of the locations. While some regions, such as Norway and Quebec utilize eco-friendly, low-carbon hydropower, others such as Poland still rely on fossil fuel sources [42].

The energy consumption of large-scale data centers poses a significant environmental challenge, alongside with the number of servers and the power required to operate them contribute substantial greenhouse emissions. As the demand for cloud services continues to grow, addressing the energy consumption and carbon footprint of data centers will be crucial to ensure the sustainability of cloud computing.

1.1 Research Objectives

The rapid expansion of cloud computing and data centers undoubtedly has raised significant concerns regarding the environmental impacts. As discussed above, the significant amount of energy usage and greenhouse emission from data centers are not fully addressed. There are several reasons to consider such as optimizing electricity consumption can lead to poor performance, using renewable energy sources may increase the costs due to limited resources.

Monitoring tools are essential for ensuring the optimal performance and reliability of large-scale computing systems like data centers, allowing administrators to track critical metrics such as CPU utilization, memory usage, network traffic, and power consumption. These tools provide valuable insights into system behavior, facilitating effective troubleshooting and maintenance. In cloud computing, Kubernetes has become the standard for container orchestration, therefore, Kubernetes-based monitoring tools are particularly valuable. Kepler [3], Kubernetes-based Efficient Power Level Exporter, is one of the most recent developed tools and considered as a potential monitoring tool to collect and analyze metrics related to resource consumption. Kepler not only provides monitoring functions, it also offers *Model Server*, which leveraging machine learning (ML) principles to enhance its capabilities.

In an attempt to explore this aspect, this thesis investigate the energy efficiency and sustainability of cloud computing environments through a comprehensive analysis of collected data from monitoring tool in a local cluster. By employing monitoring tools, the research delve into the intricate dynamics of resource utilization, workload patterns, and energy consumption. Through the evaluation of monitoring tools and power models, the thesis explores their current performance and their potential in managing large-scale computing systems.

1.2 Thesis Contribution

To study the performance of Kepler, we design a Kubernetes-based experiment to monitor energy consumption and other resource utilization metrics. The experiment is conducted on local hardware setup with customized software installations, in which Kepler is compatible with the configuration to ensure fluid operation. The experiment involved setting up Kepler to be compatible with the workstation, identifying missing elements, and the need for additional actions. To collect comprehensive metrics, we feed the cluster a Python script workload designed to interact with low-level system components. Once the script completes, we can analyze the output metrics by querying either by Kepler Model Server or a custom script. A detailed description of the experiment is discussed in Section 5.

Data is collected at small intervals, providing detailed insights into node-level performance. The output data is processed automatically both by using Kepler Model Server functionalities to categorize data into metric groups and by custom-made script. The processed data can then be used to train models, or visualized through graphs to identify characteristics.

The evaluation of the experimental results focuses on key metrics such as CPU time, cycles, estimated energy consumption, and idle nodes behavior. By analyzing the collected data, we gain a deeper understanding of the factors influencing general behavior of energy consumption. Through the experiment, we observed that the performance of Kepler is a positive sign for integrating monitoring tools to the system to optimize resource management and improve efficiency. The experiment also identify some limitations from both the monitoring tools and experimental configuration. The thesis experiment presents the study of the performance of Kepler, highlighting the potential of applying monitoring tool in large-scale computing system, as well as address the limitations to encourage further developments and improvements.

1.3 Structure of the Thesis

The remainder of the thesis is organized as follows. Chapter 2 provides the background scenery for the thesis. In this chapter, we introduce cloud computing and data centers, their services as well as common concerns. In addition, we discuss the role of energy monitoring in technology with relevant literature is mentioned, highlight the importance of efficient energy management in large-scale data centers, and provide a foundation scene of the experiment.

Chapter 3 presents a literature review on relevant technologies and topics. We discuss thoroughly from tracing tools in the operating system and hardware monitoring performance counters, to high-level, conceptual topics such as data processing in machine learning, sustainability in cloud computing, and the current states of data centers in Nordics.

Chapter 4 focuses on presenting software tools that are used in the experiment. This chapter delves into the practical aspects of these tools, and the roles in this thesis experiment context.

Chapter 5 presents the objectives of the experiment as well as the configuration. We provide the details of how each tool is configured in our machine. Moreover, we present the order and the abstract of the experiment software configuration and execution scripts.

Chapter 6 concludes the experiment results by presenting the overview of collected data, which provides insights to different metrics with graphs and numeric values. We then analyzing the data with the context of the thesis - energy consumption and performance behavior of the cluster.

Chapter 7 expands the discussion to a broader scope, where we evaluate the monitoring tools and its practicalities, determine the correlation between cloud computing and carbon footprint. Moreover, we also identify the challenge of achieving sustainability in cloud computing and presents reflections based on our findings as contributions of the thesis.

Chapter 8 summarizes the thesis by recapitulating the experiment execution and its results. Therefore suggest future research directions on cloud computing and sustainability - the primary matter in this thesis.

2 Background

This section introduces the general concepts of cloud computing and data centers. From cloud computing foundation, it dives into the energy monitoring aspect and carbon footprint from computational work. We then discuss the awareness of cloud providers of carbon emission, and how they utilize green energy sources. Finally, we present a preview of the experiment was conducted to study the behavior of a local setup cluster, and examine the energy monitoring tools used in the experiment.

2.1 Cloud computing and data centers

Cloud applications provide several services with affordable expense, from data storage services to computational server environments. The high availability to meet the increasing demand is one of the noticeable benefits of cloud computing. Builds on virtualization and allows many instances to run on a single physical server, cloud computing enhances the efficiency in resources optimization and utilization. With distributed computing as a core concept, cloud computing leverages the fundamental principles to deliver resources in a seamless way over the Internet [43]. Cloud is a great option for handling big data thanks to its scalability, flexibility and affordability. Within cloud environments, efficiently configuration of network helps manage and optimize the flow of data, enabling dynamic allocation of network resources and better traffic management. Despite these notable advantages, cloud computing also has a few drawbacks. One of the primary concerns is security. Users place their trust in service providers and use their services, but they face the risk of potential attacks that could result in data leaks.

Data center is similar to a warehouse, in which it hosts computer hardware and manage resources for computational tasks. Several big-tech companies including Google, Amazon, Meta, and Microsoft have been building their data centers in multiple continents to broaden their business in cloud. As discussed above, network plays an important role in cloud computing, therefore by having multiple data centers around the globe enables service providers approach more customers, and utilize resources in different locations.

The larger the data centers, the more complex the architecture, as they contain thousands of switches and servers. A sophisticated and optimized architecture contributes greatly to energy efficiency management. Hammadi and Mhamdi [36] discussed and compared several data center architectures, as well as the connection between energy efficiency and data center architecture.

2.2 Energy monitoring

Electricity cost represents a weighty share of the overall operational expenses in data centers. Cloud service is known for its availability, therefore data centers have no break-time and operate 24/7. In such scenarios, idle power consumption is inevitable, yet difficult to manage in order not to affect the performance of the servers. For example, from user's side the applications look like they are running, while actually aren't and cause leakage power. The reasons explain for why idle power account for a significant portion can be the interactive characteristic, low CPU workload or poor concurrency execution [13, 30]. By monitoring energy consumption behavior, cloud service providers can optimize and prevent unnecessary leakage. Not only this helps in resource allocation, but also in reducing operational costs for service providers.

On the other hand, in the pursuit of environmental conservation, governments and international bodies, such as the European Commission, have emphasized the importance of energy efficiency. Data center owners are legally obligated to comply with these regulations and must be transparent in their strategies for improving energy efficiency. These efforts not only help reduce operational costs, raise awareness about energy efficiency in data centers but also play a crucial role in minimizing negative impact on the environment. Nevertheless, energy efficiency is not the only concern in running data centers, as carbon emissions is an unavoidable consequence when operating such large-scale infrastructures.

2.3 Carbon emission in cloud computing

Global warming is certainly one of the most pressing and complex challenges facing humanity, as it poses significant risks to biodiversity, economic sustainability, and public health. In the context of cloud computing, energy and power consumption contribute considerably to global warming, primarily through the burning of fossil fuels such as coal, oil, and gas. The carbon emissions generated from energy production, heat generation, and transportation needed to support thousands of servers are substantial, releasing large amounts of greenhouse gases into the atmosphere. To address this problem, researchers have been exploring innovations in greener cloud architecture, renewable energy sources and the reuse of waste energy or heat [52, 29].

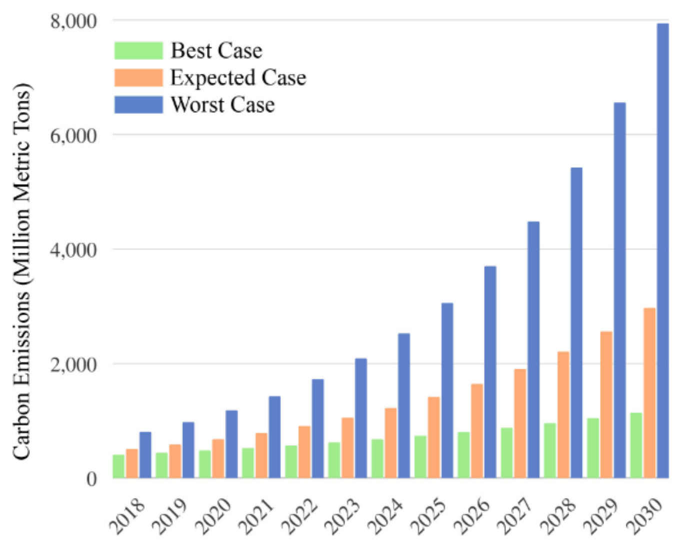

Figure 1: Estimated global carbon footprint from data centers 2018-2030 [16].

The geography factor is an important one as several studies have shown that there is a significant reduction in carbon emission from data centers in Nordic compared to other countries in Europe [25, 63]. For example, data centers in Nordic countries, where hydro power and nuclear power account for a significant share of the energy mix, produce fewer greenhouse gas emissions than those in London. Climate is another factor contributing to the differences among locations, where colder places can benefit the cooling process - a major contributor to the total energy consumption [14].

2.4 Preview of experiment

This thesis is based on an experiment designed to investigate energy consumption behavior and explore the potential of energy monitoring technologies, including Kepler—a promising prototype tool for tracking energy and power metrics in a Kubernetes environment. The experiment aimed to examine how Kepler operates within the existing setup, evaluating the strengths and weaknesses with the current version of the tool, addressing the motivation related to achieving sustainability in cloud computing goal. By feeding computational tasks to the cluster, the study provides valuable insights into the energy metrics. We analyze how Kepler gathers and processes data, along with the pre-trained models and the training pipeline of the model server, and navigate the connection between the outputs and carbon footprint. The findings from this experiment serve as the foundation for the subsequent analysis and discussions presented in this work.

3 State-of-the-Art in System Monitoring and Sustainable Data Centers

Data centers and cloud computing have become indispensable components of modern digital infrastructure, enabling businesses and individuals to store, process, and access information on a global scale. These facilities, which house computer systems and networking equipment, consume significant amounts of energy to power their operations. As the demand for computational resources continues to grow, it is imperative to explore sustainable solutions that can minimize the environmental impact of data centers and cloud computing.

As the demand for computational power continues to soar, it becomes increasingly important to explore sustainable solutions for data center operations. This chapter delves into the Operating System and eBPF analysis rationale that support Kepler, hardware performance counters and how data processed and the application of auto ML. Moreover, we also discuss the current state of research on the environmental footprint of data centers and cloud computing, examining energy consumption, carbon emissions, and potential strategies for reducing their impact on the planet.

3.1 Tracing tools in OS

Operating System (OS) is a software that manages computer hardware, software resources. OS makes use of the hardware resources and provides services to system users, as well as manages memory and I/O processes [60]. It acts as an intermediate layer between applications and computer hardware, support program executions and govern resource allocations. In cloud computing scene, OS plays an important role in managing and orchestrating resources and services provided by cloud infrastructure.

Linux is a famous and widely-used open-source OS kernel with many popular distributions, such as Ubuntu, Fedora, and Debian. The operating system offers stability, security, and adaptability across various hardware platforms. Linux is open-source nature, therefore developers can contribute to its continuous improvement and innovation. Moreover, Linux also power smartphone operations, as it is the underlying kernel for several mobile OS. Android is a notable example, where most of smartphones in the world are developed with it as the OS. Linux kernel is optimized for performance, allowing users to manage resources efficiently, explore the capabilities of their smartphone further compared to other mobile phones OS.

Large-scale computer systems such as data centers would require huge resources to report and troubleshoot technical problems. In previous section, we stated that hyper-scale data centers are hosting thousands of servers and other hardware equipments to provide 24/7 services to customers, hence it is expensive to detect bugs, network issues, or fix hardware malfunctions. The cost to address the problems can be higher than expected if there is a delay in detecting those issues, or due to unprepared plans. During the era of big data dominance, data centers can support various jobs from business side to more computing jobs such as ML trainings, and all of these jobs demand varied allocation of resources. Analytics jobs in business field usually require CPU resource, while ML tasks are more memory-hunger [64]. As a result, a fair solution for resource allocation and scheduling is a must to ensure service performances as well as managing operational costs.

Tracing is a reliable method of collecting information of a system for deep analysis and workload anticipation. Especially in data centers, manually monitoring systems would require significant human involvement, which consequently increase expenses and is more time-consuming. With tracing tools, engineers can automatically monitor and alert system performance, identify potential issues, and optimize resources, saving both time and effort. Tracing data provides comprehensive insights into the operation of applications and the flow of programs [40]. Linux operating system supports several tracing tools such as LTTng [27], SystemTap [8], BPF/eBPF that can trace low-level operations, including system calls and kernel events, or Microsoft Windows OS with DTrace [35]. Except for eBPF, the others tracing tools have been in the market for quite some time. BPF was introduced shortly after the release of Linux OS in 1991, followed by DTrace in 2003, and SystemTap and LTTng are respectively released in 2005.

3.1.1 DTrace

DTrace is short for Oracle Solaris Dynamic Tracing Facility, designed to be the observability technology that helps users understand the behaviors of system and applications in development and production [35]. It is a command-line tool enabling users to monitor and troubleshoot system performance in real-time, as well as allowing dynamic instrumentation of both the kernel and user-space code without interfering the original scripts [15]. Cantrill et al. [15], the original authors of DTrace, introduced several other features of the tools that are beneficial to users such as data integrity, scalable mechanism for aggregating data, or that DTrace is virtualized per consumer, i.e. no limit on the amount of concurrent consumers. Moreover, it supports many data processing and analysis methods such as aggregations, histograms.

One notable advantage of DTrace is that it requires minimal programming skills. With a variety of pre-made scripts available, DTrace is accessible to users regardless of their programming experience [35]. For users who are familiar with programming in C, they will get used to DTrace easily thanks to the similarity between C programming language and D programming language-a C-style language that incorporates all ANSI C operators and enables access to the kernels's native types and global variables.

While DTrace is powerful and one of the earliest tracing tools for OS, it still contains some limitations. For instance, it is mainly specialized for Solaris and has limited compatibility with Linux OS. As Linux become one of the most common OS for software development, and especially for cloud infrastructure, this lack of support has made DTrace less attractive compared to other tracing tools such as SystemTap or BPF/eBPF.

3.1.2 SystemTap

SystemTap is an infrastructure that monitor performance and kernel troubleshooting for Linux [51]. It is designed to support application in a efficient and robust manner, with the goal to shorten time and complexity for diagnosing kernel activity issues. Other tools such as *netstat, ps* serve the same purpose; however, SystemTap has a broader scope as it contains "more filtering and analysis options" [8].

SystemTap seeks to enhance the set of Linux monitoring tools by offering users the infrastructure to observe kernel activity, while integrating flexibility and ease-to-use features. With flexibility feature, SystemTap allows users to implement scripts for monitoring kernel actions, system calls and other events in kernel space [7]. These scripts can be used to extract and filter data, contributing to analysis process. Moreover, the ease-to-use helps users enable monitor scripts without having to reboot and recompile the kernel.

Rocca [55] has conducted a comparison between SystemTap and bpftrace, where the author observed that the foundation kernel infrastructure of both tools is similar, and both facilitate instrumenting the user-space and kernel through scripts. However, there are some key aspects that make bpftrace more popular such as it is generally faster than SystemTap, offers many facilities for straightforward aggregation and reporting that help simplifying the process of extracting data. SystemTap, on the other hand, requires users to install additional dependencies in order to fully utilize its capabilities. This can increase the complexity of development and maintenance, especially in environments with strict package management policies.

3.1.3 LTTng

Linux Trace Toolkit next generation, shorten as LTTng, is an open-source tracing framework for Linux, designed for monitoring and diagnosing the performance of kernel and user-space applications. It depends on the Linux Kernel Markers as the main information gate to inject instrumentation to the kernel [26]. The design of LTTng allows users to trace the kernel and user-space at the same time with minimal impact on the traced system. Created by Mathieu Desnoyers and Michel Dagenais [27], LTTng is a low-overhead tool written in native C with LTTng-UST library for tracing user-space applications [21]. It is based solely on atomic operations, allowing LTTng to trace many code sites to even kernel code.

Considered as a powerful tool in Linux environment, LTTng has some disadvantages making it less captivating. Firstly, the setup and configuration can be complex, especially in distributed systems or when involving kernel modules. Secondly, LTTng lacks built-in visualization abilities, relying on external tool which can be expensive. Being a Linux tool also limits its compatibility, for example, Windows OS, which is the dominant OS for decades, may not suitable and LTTng can not be installed. The complexity and limited compatibility can pose challenges for beginners.

3.1.4 BPF and eBPF

Berkeley Packet Filter, know as BPF, is a packet filtering tool, first introduced in the BSD operating system [47]. The mechanism of BPF includes two primary components: the network tap, which gathers copies of packets from the network drivers; and the filter component, which determines whether a packet is accepted or rejected, as well as the amount to send to the monitoring application. BPF was later introduced in Linux OS in 1997, where it was used in the tcpdump utility as an efficient method for capturing the packets to be traced [54]. McCanne and Jacobson are the authors of the first original documentation of BPF [47]. As the complexity of the modern systems grew, so did the need for more flexibility and programmability within the kernel, eBPF as an evolution come to the world, as an expands of the original concept by enabling the execution of a wide range of programs directly in the kernel, while maintaining minimal impact on system performance.

Extended from BPF, eBPF was introduced in kernel version 3.18 in 2014 with some notable improvements. The eBPF [1] program is a new technology origins from Linux kernel, allowing sandboxed program to run with privileged mode, i.e. within the operating system. Additional capabilities such as the updates of the interpreter, *bpf()* system call become available in user-space, eBPF maps were included which play an important role in transferring data between programs and allow sharing between the kernel and user space [59], and many new BPF helper functions [54] have contributed to the comprehensiveness of eBPF compared to other tracing tools. Moreover, the eBPF verifier was introduced, which is the key element to enhance safety when running eBPF programs. Programs will be verified at loading phase to avoid crashes and vulnerabilities in the kernel, ensuring memory access is safe.

There are multiple eBPF-based projects for networking, observability, and security thanks to the innovation of the technology. When an eBPF program is loaded into the kernel, it will be verified to make sure that the program have a limits of instructions to prevent infinite loops or resource attack. The root access is needed if ones want to disable the verifier, which helps avoid malicious eBPF programs that can attack the kernel memory outside the scope of the programs [56]. With this, the OS ensures safety and execution efficiency, with high-performance applications [1]. As cloud computing pivoting in response to evolving technical trends, eBPF becomes an excellent tool in providing transparency of every programs on the host device. Some have predicted that, for example Gartner [33], more than 95% of modern digital workloads will be migrated to cloud-native environments. Concurrently, Kubernetes is becoming the key standard for cloud orchestration and cloud-native architecture. With more kernel subsystems becoming extensible through eBPF, drivers and kernel modules can be implemented in eBPF in near future. One of many advantages of eBPF is the integration of the *XDP* and *AF_XDP*, which aligns seamlessly with Kubernetes [59]. Leveraging this advantage, Kepler - Kubernetes-based Efficient Power Level Exporter - trace and collect metrics utilizing performance counters with high accuracy.

Kepler [3], an open-source tool for exporting power levels in Kubernetes environments, utilizes eBPF to probe CPU performance and provide deeper insights into power and energy consumption behavior. The collected data is available in Prometheus [6], which users can interact through GUI to retrieve datapoint, or apply Kepler Model Server query function. This integration allows Kepler to efficiently monitor and analyze resource usage, enabling better power management and optimization within Kubernetes clusters.

To recap, BPF and eBPF represent transformative tools in the evolution of Linux, offering unparalled flexibility and performance. The continuous integration into modern systems broaden new possibilities for innovation in areas such as networking, observability and monitor, and sustainable computing.

3.2 Hardware monitoring by performance counters

As computer hardware becoming advanced with the evolution of technology, the gap between hardware-level operations and software performance analysis also clearer. To address the matter, hardware performance counters (HPCs) were introduced in order to help researchers and developers to understand the complexity of computing systems, face the challenges in optimizing software application performance. Before HPCs, there were traditional profiling tools; however, they are insufficient for collecting fine-grained, real-time data.

HPCs are specialized registers build into modern processors, which can be used to monitor and measure many low-level events and activities at the CPU. By providing a detailed view of hardware behavior, HPCs serve as invaluable tools for performance profiling, debugging, and optimization of applications. Their integration with advanced frameworks, such as eBPF, further extends their utility, enabling efficient performance analysis in dynamic and complex computing environment.

Although performance counters are expected to aid in the analysis, debugging process, Das et al. [22] reported that over "45% of the papers in application areas that include performance analysis and optimization" do not value HPCs highly. The authors also stated some common reasons for this problem are the inconsistency in performance counter results, and the limited portability of HPCs events. Some events can be difficult to validate effectively as they depend on precise knowledge of the underlying CPU architecture, as well as effect of external timing factors that are beyond user's control. For instance, program schedules, memory allocation, OS tasks, and multi-processor interactions are subjected to change.

By utilizing HPCs, many tools are developed to contribute to the performance measurements. However, the results are varied even the measurements are for the same task. Moreover, performance counters may overestimate certain events on specific processor. Weaver et al. [65] reported that on Platinum D processors, the instruction-retired event can be overcounted in cases such as lazy floating-point handling by the OS, or the execution of specific instructions like *emmes*, *fldcw*.

Hardware performance counters are essential tools for understanding the behavior of processors and optimizing system performance. They offer detailed insights into processor events such as instruction execution, cache usage, and memory accesses. There are also challenges using HPCs besides the mentioned advantages. Issues such as non-determinism, platform-limitation, and overcounting should be addressed for more accurate and reliable performance analysis. In conclusion, HPCs are still essential for optimizing modern computing systems, especially in high-performance environments.

3.3 Data processing for ML pipelines

Data processing is a fundamental step in any machine learning pipeline. It involves transforming raw, unstructured data into a suitable format for model training and evaluation. The initial step is data ingestion, which entails collecting data from diverse sources such as databases, APIs, or files. The data is then converted into a standardized format, like CSV, JSON, or Parquet, for further processing.

Data cleaning and pre-processing are crucial for ensuring data quality. This involves removing inconsistencies, errors, and missing values, as well as applying techniques like imputation, normalization, and outlier detection. Feature engineering involves creating new features from existing ones to enhance model performance. This may include feature selection and dimensionality reduction techniques. Once cleaned, the data is transformed into a suitable format for machine learning algorithms. This involves feature engineering, normalization, and scaling. Feature engineering creates new features from existing ones to improve model performance. Techniques like one-hot encoding, feature scaling, and polynomial feature expansion are commonly used. Finally, the dataset is divided into training, validation, and testing sets to evaluate model performance and prevent overfitting.

Challenges in data processing include data quality issues, handling large and diverse datasets, and addressing data privacy and security concerns. Popular tools for data processing include Python libraries like Pandas, NumPy, and Scikit-learn, SQL, and Apache Spark. Cloud platforms like AWS, GCP, and Azure also play a significant role in data processing pipelines. By effectively addressing these challenges and utilizing appropriate tools and techniques, organizations can unlock the full potential of their data and build robust, accurate machine learning models.

Kepler Model Server is developed to help simplify data processing, allowing users to build effective models with less complexity. From raw output data, Kepler Model Server offers *isolate* and *extract* methods to process data, where *isolate* is for separating idle power from absolute power to compute dynamic power, and *extract* step is for cleaning up the data separately for each feature group [3]. From here, users can continue with *train* or *train_from_data* to train one or many models at once. According to Kepler documentation [3], DynPower models are trained with isolated dynamic power, and AbsPower models use absolute values. As DynPower models do not include idle power, AbsPower models are used for node power estimation.

3.4 Cloud computing and the sustainability

Recent years have seen a dramatic integration of networks, computing, and the internet into people lives. With billions of users and devices actively engaged in computational tasks, a more robust and efficient infrastructure is essential. Cloud computing, supported by extensive data centers, has emerged as a leading platform for delivering these services. The rapid development of technologies such as generative AI, cloud-based gaming, and cryptocurrency has led to increased demand for large-scale computing. Data centers, operating 24/7 and equipped to handle peak workloads, require substantial amounts of electricity to power their operations. Alongside with the exceptional growths, other elements are relevant and contribute to the impact on environment as well.

One of the most notable factor is energy consumption, which has become a major concern in the computing industry, particularly in large-scale computing environments like data centers. These infrastructures consume vast amounts of energy, significantly contributing to environmental impact. Their reliance on fossil fuel-powered electricity generation contributes substantially to carbon emission. In 2006, data centers in the United States consumed 1.5% of the national's total electricity, equivalent to the energy demand of 5.8 million households. Globally, data center electricity usage contributed to approximately 6% of total CO2 emissions in 2006. According to The Climate Group [34], data centers emitted 116 million metric tons of CO2 in 2007. Despite technological advancements, the greenhouse emissions crossed 259 MtCO2 in 2020.

Many prominent service providers have advocated and documented their approaches to optimizing data center energy efficiency and environmental sustainability. By decreasing energy consumption, organizations can achieve significant operational cost savings and indirectly lower carbon footprint. Nevertheless, some research suggests that energy efficiency initiatives may not always correlate with carbon emission reductions [31]. Beyond server power consumption, cooling and air conditioning systems are major energy consumers, accounting for up to 30% of total energy usage [53]. The rapid escalation of data center energy consumption has compelled operators to pursue energy-efficient solutions to mitigate both financial and environmental costs [62], including the adoption of advanced cooling systems and facility designs.

Cooling is an important factor in operating data centers as with intense computational loads, devices can generate a significant amount of heat which can lead to negative impacts on the operation. Traditionally, data centers are kept in cool temperatures, for example in the range of 18 - 27 Celsius degrees [61], and to maintain such stable temperatures requires a tremendous amount of energy. Cooling systems remain as one of the foremost challenges in data centers. Many researches have been done to optimize the cooling system, including research on position of racks in data centers [50], proactive thermal management to prevent heat imbalances [41].

In addition to air cooling method, liquid-cooling is a notable method as it reduce total energy consumption compared to air-cooling. Zhang et al. [66] conducted a comprehensive survey on data center cooling systems with many popular cooling strategies. According to the survey, new technology trend-edge computing-can increase complexity in improving cooling system. Despite that adopting edge in data center can help reduce transfer latency and network overhead, the nature of edge computing requires cooling systems to cover smaller units. Overall, energy consumption and CO2 emissions can be substantially reduced through improved cooling management. By utilizing climatic conditions and advanced technology, data centers can expand while minimizing environmental impacts.

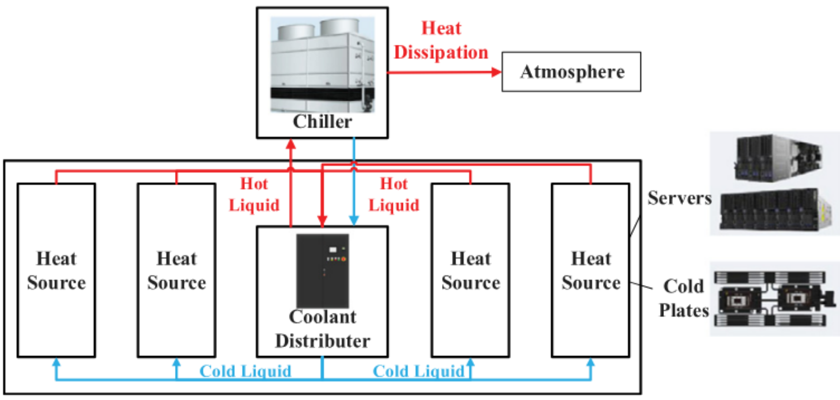

Figure 2: Indirect liquid-cooling system [66].

3.5 Data centers in Nordics

The rapid growth of digital technologies and the increasing demand for data processing and storage have led to a surge in the construction and operation of data centers worldwide. These facilities, often housed in massive buildings, consume significant amounts of energy to power their server and cooling systems. Consequently, the environmental impact of data centers has become a growing concern. National regulations and global governance prioritize expanding information infrastructure while the energy consumption and environmental impact are more vital. Corcoran and Andrae [20] observed that there is a growing trend to shift energy consumption towards network and data centers infrastructure, where costs are less visible to users. As data centers consume a significant amount of energy, the sources of electricity also contribute to the carbon emission.

To mitigate this impact, several regions, including the Nordic countries, have emerged as attractive locations for data center development thanks to their favorable climatic conditions and abundant renewable energy resources. In addition, Nordic countries have stable political and supportive government that support businesses to open data centers there, as well as a secure operating environments with low possibility of attacks. For example, Iceland's government publishes their policy to attract "green investments", including low tax, renewable energy, and sustainable management of natural resources and land [37].

In Finland, there is analysis revealed that capitalizing on waste heat could generate substantial lifetime savings exceeding millions of euros over a 20-year lifespan. The most economically advantageous approach involved employing a heat pump (HP) to elevate waste heat temperatures to 75°C and subsequently selling it to a district heating (DH) network, either on the supply or return side, contingent upon external temperature conditions. Although the initial expenditure for HPs capable of attaining 75°C (with a

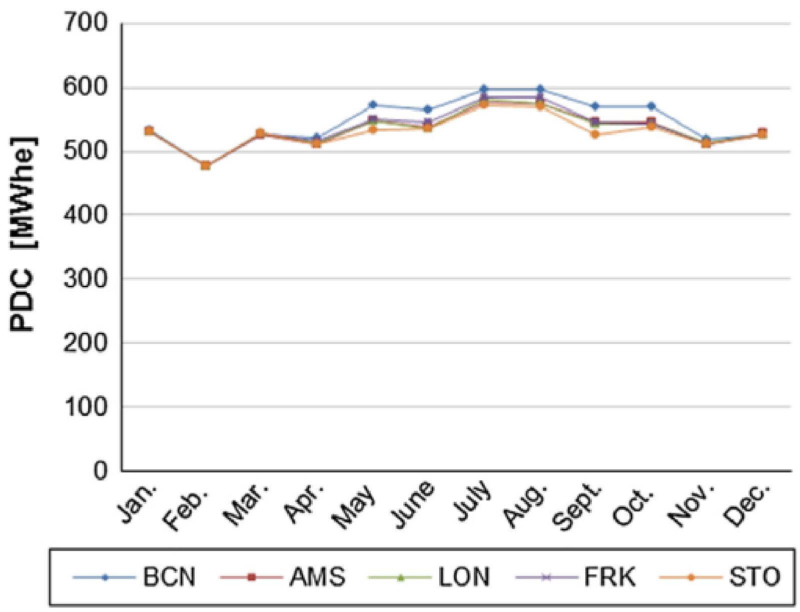

Figure 3: Monthly electricity consumption in different locations in EU [24].

coefficient of performance of 3.5) surpasses the baseline scenario utilizing free cooling without waste heat recovery by €420,000, the annual revenue derived from selling waste heat to the DH network would approximate €600,000. This implies a payback period of under two years, as the income from heat sales would surpass the augmented electricity consumption associated with HP operation [63].

With geographical advantages in the Nordic regions, data centers can make use of free cooling methods to reduce energy consumption and operational costs. The low ambient temperature can aid in the need for energy-intensive air conditioning systems, therefore minimize environmental footprint. For example, Jyväskylä, Finland has average temperature below 5 Celsius degree over half of the year, while other central Europe countries have shorter period [63]. Moreover, with generous sources of renewable energy such as hydropower and nuclear power, Nordics offer low energy prices compared to others.

According to [63, 19], the Nordic countries are ideal for data centers because of their cold climate, affordable electricity for industries, and the potential for efficient cooling. In many areas, temperature management can be effectively achieved not just by the natural cool temperatures but also by utilizing nearby natural resources. For instance, cold and dry weather may require additional humidification, which can be sourced from seas and lakes to meet the minimum inlet humidity for data centers. More over, Depoorter et al. [24] also emphasized the importance of data center locations. In Figure 3, the authors show that during the summer period, Stockholm consumes much less than other locations, for example Barcelona that have the highest values.

The study [24] highlights significant regional differences in energy supply, carbon emissions, and costs for data centers. Stockholm is considered the one of the most eco-friendly, greenest locations thanks to the use of clean energy sources, while London is using coal-heavy electricity grid. According to [63], a data center in Stockholm would produce over 30 times less carbon emission than one in London. Nevertheless, the authors note that the comparison is debatable due to the impact of international electricity transmission connections, which also affect the validity of the statistic.

In general, by leveraging its natural advantages, progressive policies, and strong commitment to sustainability, the Nordic region is well-positioned to maintain its figure as a global pioneer in sustainable data center development. As the world increasingly relies on digital technologies, the Nordics' efforts in increasing environmental sustainability will play a vital role in powering the digital economy.

4 Experimental tools

This section provides details on the software applications and libraries utilized in the experiment, highlighting their roles in the experimental setup. Additionally, we address the carbon footprint issue associated with computational development, a key motivation for this paper.

4.1 Kubernetes

Kubernetes (K8s), an open-source platform originally developed by Google and now maintained by the Cloud Native Computing Foundation (CNCF), has emerged as a dominant force in container orchestration. It automates the deployment, scaling, and management of containerized applications, offering a robust and scalable solution for modern distributed systems.

Unlike Platform-as-a-Service (PaaS) offerings, Kubernetes operates at the container level, not directly managing the underlying hardware infrastructure [4]. This distinction allows for greater flexibility and portability, as containerized applications can be deployed across diverse hardware environments. K8s leverages containerization technologies like Docker, grouping individual containers into logical units called Pods for easier management and service discovery. Kubernetes is self-healing, supports horizontal scaling, provides built-in service discovery and load balancing, and facilitates automated updates. It securely manages sensitive information and includes a master node with components like the API server, controller manager, scheduler, and etcd, as well as worker nodes running the Kubelet, Kube-proxy, and container runtime.

Kubernetes is an open-source platform designed for automating deployment, scaling and management of containerized applications. It orchestrates clusters of machines, known as nodes, to manage the life-cycle of containers, ensuring efficient resource utilization, scalability and high availability. Kubernetes (k8s) helps managing cloud-native applications infrastructure with automated deployment and scaling, assist developers in handling and maintaining the efficiency and reliability of the applications.

4.2 Kepler

Kepler (Kubernetes-based Efficient Power Level Exporter), a Prometheus exporter, utilizes eBPF program integrated into the kernel's pathway, which can be used to probe CPU performance and data related to power and energy consumption behaviors. The tool provides various metrics, including energy consumption of CPU, DRAM, GPU, which can be exported to and stored in Prometheus. Kepler can perform fine-grained monitoring of power usage across different components if it can access RAPL metrics [3]. It also have model server, which is essentially a ML engine designed for power consumption estimation by training collected data. Currently, the model assists Kepler in choosing the most suitable power model based on factors like CPU type and specific needs. The long-term goal is for Kepler to independently select the best model, replicating the Model Server's logic. For example, if RAPL energy source is not available, Kepler will use its model server to estimate the metrics. Server power consumption is the energy used by a server's physical components (such as the CPU, memory and storage) to execute software applications, usually measured using integrated power meters within the server. However, the consumption varies even with identical machines with the identical workload, which makes the modeling more difficult. The power consumption includes the idle state and the working state, and given the idle state is often static, the workload power is extracted by deducting the idling power consumption from the total measured power, also known as profiling approach [10]. This approach to train the power model exhibits certain limitations in cloud environment. The lack of access to important data points, such as CPU frequency and co-located virtual machines, limits the accurate application of matching profiles. Furthermore, the dynamic and noisy nature of background process power consumption complicates the container power modeling, particularly when relying on static profiling method [17].

Kepler team introduces a pipeline framework of power model server, which collects real-time, "fine-grained energy-related metric exporter on Cloud" [17]. Kepler model server is designed to train power models, can be either deployed independently or accompanied with Kepler to find the most suitable power model for a given environment, ensuring accurate energy consumption analysis. The logic behind the model server is expected to apply to Kepler in the future in order to help it select power model effectively and accurately. The pipeline for training the model server is designated for both server and individual containers. According to the develop team [17], the pipeline facilitates the training of power models, leverages usage metrics available through the exporter. The developers aim to emphasize a power isolation module, a critical step for separating the measured server power consumption into the power usage attributed to each container's resource consumption. Once the training is finishes, the model operates independently from platform-specific data or power measurements.

4.3 Prometheus

Prometheus [6] is an open-source monitoring and alerting toolkit designed for reliability and scalability, primarily used to record real-time metrics in a time series database. Originally developed by SoundCloud, Prometheus has become a crucial part of the observability stack in modern cloud-native environments. It excels at collecting, storing, and querying metrics from various sources, making it particularly well-suited for monitoring dynamic, containerized environments like those managed by Kubernetes. Within K8s environment, Prometheus is used to monitor cluster performance and resource utilization, exporting detailed insights into the behavior of applications and infrastructure.

Together with Kepler, they collect data about power consumption by leveraging kernel advanced technology, store the data and perform analysis, identify pattern of usage, therefore help developers measure and optimize the workload.

4.4 Grafana

Prometheus itself also provides illustrations, like graphs, based on the collected data; however, there are limits in it features, such as the graphs are more ad-hoc oriented, which is not suitable for long-time uses. Grafana [2] is a bright partner as it can retrieve data directly from Prometheus source. Grafana offers extensive features for illustration, including templates for dashboards, different types of graph, chart.

For example, Grafana enhances collaboration and communication within teams. Dashboards can be easily shared and accessed by team members, providing a common ground for discussions, trouble shooting, and decision making around system performance and operational insights. Moreover, it facilitates proactive issue detection and resolution. By setting up alerts for anomalies and potential problems, teams can be quickly notified via various channels, enabling real-time response and minimizing downtime.

In this experiment, there are two Prometheus data sources, one contains Kepler metrics, the other contains Kafka data. Grafana can handle this nicely since it support multiple sources to build dashboard. We can easily select which data source to use for each panel within the dashboard, and Grafana will seamlessly fetch the data and generate corresponding graphs.

4.5 Apache Kafka

Apache Kafka is a distributed messaging system designed to collect and deliver high volumes of log data with low latency. In this experiment, Kafka is used to transport data to worker pods running in Kubernetes. Kafka, an Apache open-source project, is widely adopted by various large tech companies, including those in manufacturing, banking, and telecom sectors. In Linux or UNIX-like environments, Kafka efficiently delivers bytes in a log segment file from a broker to a consumer using the *sendfile* API.

Kafka's architecture offers scalability without downtime, and its partitioning feature is ideal for handling large volumes of data. Messages are organized into different topics, and consumers can subscribe to any topic by publishing a message to it. When a topic becomes crowded with messages, the partitioning strategy divides the topic into smaller partitions, treating each as an independent topic. This partitioning, along with other features like replication, distribution, and fault tolerance, makes Apache Kafka highly reliable. With such popularity, Apache Kafka provides frameworks and extensions for a wide range of programming languages, making it easier to adapt to several tech stack for message queue. The administrative clients can also create APIs that streamline topic creation, manage partition and replication factors.

5 Experiment: Objective and Configuration

This section presents the detail regarding the configuration and execution of the experiment. Thanks to Jaakko Harjuhahto and John Wickstrom's equipment design, we inherit the set-up and remain most of the configuration as it was. Hardware components including a master node and 5 worker nodes connecting together through a switch; Linux is the Operating system (OS). From the novel point of view, we aim to utilize the set-up to run Kepler and measure the energy metrics while feeding customized workloads. Moreover, this experiment also serves as a study on Kepler as a prototype, which provides a concrete view on this monitoring tool.

The scope of the experiment is limited to a certain extent as the facilities are incommensurable. To run Kepler with the current set-up, platform energy source is excluded as the lack of support from the hardware, i.e. ACPI (Advanced Configuration and Power Interface). The result data should be regarded as estimates, due to variations in hardware element consumption and energy source, which may influence the overall outcomes. The final analyzed data is expected to provide conclusions regarding energy consumption patterns and their correlation with the carbon footprint of cloud computing.

5.1 Machine configuration

5.1.1 Linux and eBPF

Linux provides a flexible platform for developing and deploying necessary tools. Leverage the powerful capabilities of eBPF technology, the cluster is enabled the precise tracing of system calls and kernel events, providing granular insights into energy consumption patterns. In particular, eBPF is a crucial tool for tracing system behaviors and kernel events with security, allowing for detailed analysis of energy usage patterns and the identification of potential malfunctions or optimization opportunities.

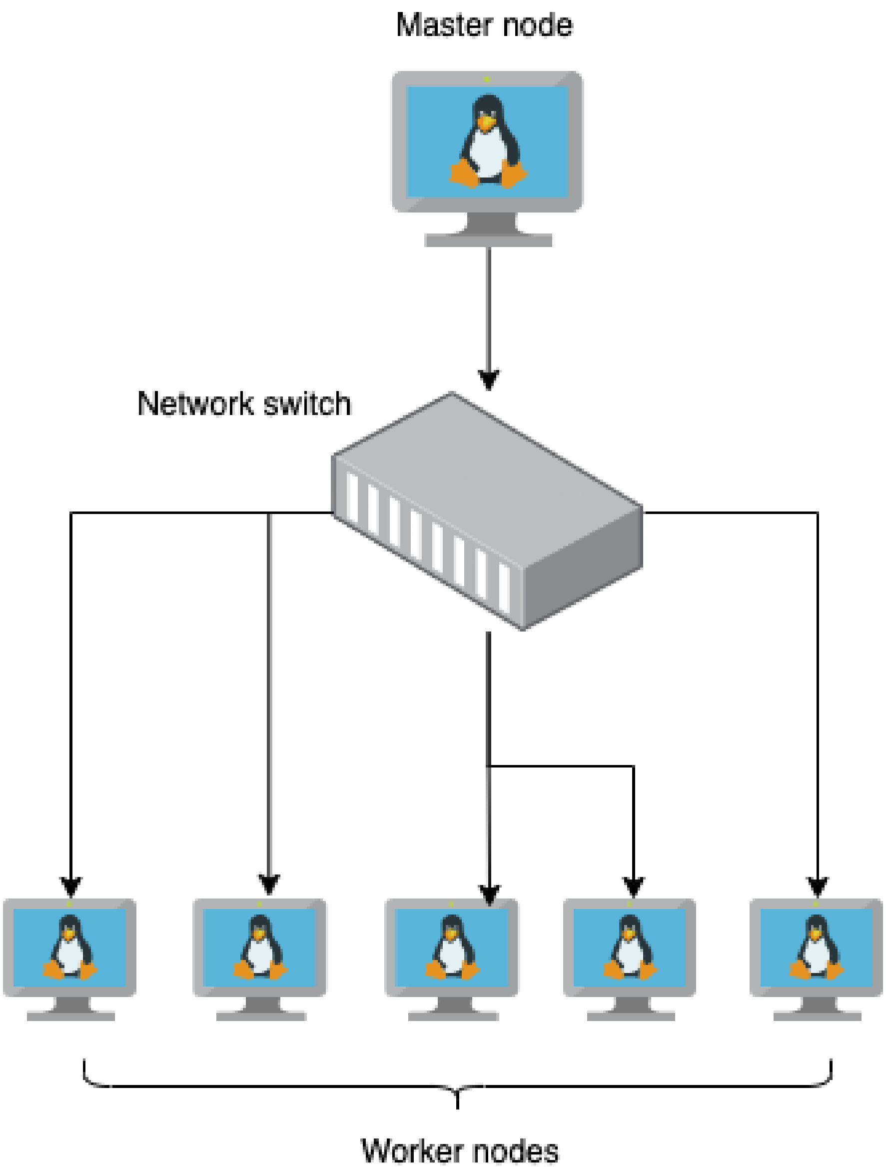

Figure 4: Local hardware setup by John Wiström and Jaakko Harjuhahto.

Figure 2 illustrates the local setup, which includes a master node run in Ubuntu, and five worker nodes. They are connected through a network switch, ensuring that commands from master node will be passed to worker nodes.

5.1.2 RAPL

As the awareness of energy consumption and environmental impacts has been rising, people started to measure computing hardware energy consumption, especially data centers, which perform thousands of computational executions 24/7. Previously, many researches on power estimation model were published [48, 44]; however, the accuracy and granularity are not ideal, as the quality of the model plays the key role in the measurement, which can be affected by the inconsistency nature of workloads. Eventually hardware vendors started to provide power metering feature, which support better the power consumption measurement.

RAPL (Running Average Power Limit) [23], developed by Intel, is a hardware feature allows user to monitor energy consumption across different domains of the CPU chip, attached DRAM, and on-chip GPU with promising accuracy. It is possible to programmatically get real-time data on the power consumption of the CPU package and its components, as well as of DRAM memory that the CPU is managing. RAPL provides raw data about the overall power consumption of the system, which are crucial in the effort of understanding the behavior the cluster [38]. According to Khan et al.[38], there has been several research done to measure the accuracy and performance of RAPL overtime [58, 28, 39]. The common result is that RAPL eventually improve and perform better throughout releases. Scientists compare RAPL readings to highly precise external power meters to assess this accuracy. By creating a model, they can see how closely RAPL aligns with the overall power consumption measured by these external meters [38].

By taking the advantages of the feature, Kepler can collect data of different power domains. It correlates RAPL's overall power consumption data (data source) with the container-specific resource utilization information obtained via eBPF. If RAPL is not available, Kepler will use its model server to estimate [3].

5.2 Software Configuration

The goal of the experiment is to study the energy/power consumption behavior of the cluster as in cloud native environment, applying power monitoring tools like Prometheus and Kepler. The nodes are fed with workload including ML models (YOLO), or applications that interact with core hardware components, particularly the CPU and cache memory. Applications that are computationally intensive or exhibit irregular memory access patterns tend to place a considerable load on these resources, impacting overall system efficiency and energy consumption.

5.2.1 Kubernetes

As introduced in Background section, Kubernetes is utilized in this experiment as a mean of scaling and managing services, orchestrating the deployment environment. By using Kepler, a Kubernetes based application, it is essential to set up the Kubernetes environment that support and compatible with the workload applications. The local hardware station configured with Kubernetes allows for the replication of real-world scenarios where services need to scale elastically based on demand, approximating the behavior of cloud-based infrastructure. Kepler provides insights into the energy consumption and performance metrics of the Kubernetes cluster, which are essential for fine-tuning the application workloads and analyse the energy consumption behavior. The correct setup and configuration of the Kubernetes environment are critical to ensure compatibility with the workload applications being tested. This includes configuring the cluster to support the specific requirements of the applications, such as resource allocation, scheduling, and performance monitoring, while maintaining seamless interaction with Kepler.

5.2.2 Kepler

With this experiment, we want to examine the energy consumption of the clusters, such as the operation consumption, or the amount of energy for starting the clusters. The carbon footprint can be estimated based on the general data of the consumption, but it is difficult to estimate at a more detailed level (component levels). While this experiment provides valuable insights, the scale is intentionally limited compared to real-world data centers, which are our primary area of interest. This initial step lays the groundwork for extending our research to the broader data center environment.

The experiment studies Kepler as a prototype tool

5.2.3 Cluster configuration

Install dependencies
The cluster is configured within an *Ubuntu environment*, making it crucial to install the appropriate packages and dependencies to ensure the experimental setup is properly established. Key components such as *Docker*, *Kubernetes*, and *Golang* are installed, alongside other essential command-line tools, to facilitate the smooth operation and management of the system.

The installation is done once when setting up the cluster. There are scripts for automating this process, which can be found via the Github repository (add repo link).

Launch Kubernetes clusters
The provided scripts automate the connection process between the *master node* and the *worker nodes*. Initially, *Kubernetes (K8s)* is set up on the master node. Following this, the script configures Kubernetes on the worker nodes and automates the connection of the five worker nodes to the master node

using a newly generated token. This process ensures seamless integration and communication within the cluster.

Deploy monitoring systems

In a Kubernetes environment, *kube-prometheus* includes Kubernetes manifests, Grafana dashboards, and Prometheus rules, along with documentation and scripts, to facilitate straightforward end-to-end monitoring of Kubernetes clusters using Prometheus and the Prometheus Operator. The system is designed to enable close monitoring of the cluster's resources, with Prometheus responsible for scraping and temporarily storing metrics data, while Grafana provides near-real-time observation of these metrics through customizable dashboards.

The first step involves cloning the repository and applying the Kubernetes configuration. Subsequently, a script is executed to manage the remaining setup tasks, including the configuration of Grafana. Once initialization is complete, both Grafana and Prometheus will be accessible within the Kubernetes cluster. Port forwarding is then configured to allow access through the master node's localhost.

The next script automates the deployment of *Kepler*, a monitoring tool, on the Kubernetes cluster. It starts by cloning the Kepler repository from GitHub and navigating into the cloned directory. The script then installs the necessary tools and generates deployment manifests, offering options to deploy Prometheus, enable high granularity in metrics, and deploy an estimator. Additionally, it deploys Kepler modules on each cluster node to track energy usage and sets up a Kubernetes Metrics Server to enable dynamic scaling of pods. After generating these manifests, the script applies them to the Kubernetes cluster. Finally, it monitors the status of all pods across all namespaces, ensuring that the installation completes successfully.

Deploy Kafka

Kafka is used to transport data to worker pods running within the Kubernetes environment, making it essential to monitor Kafka's performance to assess experimental outcomes accurately. *JMX (Java Management Extensions)* scraping is employed to collect Kafka metrics from Docker containers, with the resulting data stored in *Prometheus* and visualized using *Grafana*. JMX provides valuable insights into the functioning of cluster hardware, as well as the performance of Kafka topics and partitions.

To collect Kafka metrics from Docker containers, JMX scraping is used. The collected metrics are stored in Prometheus, and Grafana visualizes the data. These insights are crucial for evaluating hardware interactions with the clusters and understanding the efficiency of Kafka topics and partitions.

Although Kafka is typically deployed within Kubernetes, the current experimental configuration encounters conflicts between some of the integrated tools. As a result, Kafka and an additional Prometheus instance run outside the Kubernetes cluster. After each experiment, data from both Prometheus instances—one monitoring Kubernetes and the other monitoring Kafka—is aggregated and analyzed to provide a comprehensive view of system performance.

In addition, *Zookeeper* is installed and used to manage and coordinate the Kafka nodes, ensuring high availability and fault tolerance. Zookeeper detects node failures and facilitates recovery, contributing to the resilience of the Kafka system. Together, Kafka and Zookeeper form a highly complementary pair, widely adopted in various projects for ensuring system stability and efficiency [18, 49].

5.2.4 Feeding workload

To study the behavior of the cluster, we customize workloads that utilize cluster resource, and interact with hardware components. The tasks are designed to perform various resource-intensive tasks, specifically targeting CPU and memory operations, to simulate different workloads. The main functionalities of the script can be summarized as follows:

CPU-Intensive Task: A function that executes computationally heavy operations for a defined duration by performing repeated calculations (squaring numbers) to stress the CPU.

Memory Allocation Task: This function allocates large arrays of memory (in MB) over a specified duration, simulating high memory usage. The allocation rate can be controlled by adjusting the sleep time between allocations.

Memory Access Pattern Task: This task simulates different memory access patterns (sequential and random) by writing random values to arrays. The sequential pattern accesses memory in order, while the random pattern accesses memory at random indices, both over a specified duration.

Matrix Computation: A function performing matrix multiplication on two randomly generated matrices of size 10,000 x 10,000, representing a typical computation-heavy workload.

Sequential and Random Data Access: These functions simulate sequential and random data access patterns in a large array, highlighting differences in cache performance. Sequential access promotes cache hits, while random access is likely to cause more cache misses.

The script's main function orchestrates the execution of these tasks sequentially. It runs the CPU-intensive task, followed by memory allocation, memory access patterns (both sequential and random), a large matrix computation, and then sequential and random access tasks. Each operation is logged to indicate the start and completion, facilitating monitoring of resource usage during these tasks.

This script is a benchmark tool to simulate and analyze the performance of CPU and memory usage, examining cache behavior, memory access patterns, and computational workload efficiency. It offers a practical framework for studying system performance under varying resource demands. By creating Docker image with the workload script, we then deploy and the workload is now run inside Kubernetes environment, which is crucial for Kepler measurement.

5.2.5 Execution

The initial phase of running the experiment was primarily focused on testing workstation setup and ensuring the smooth operation of Kepler. During this period, we observed that the cluster occasionally froze, and Kepler was not responding as well. By adjusting the speed of execution and some small configurations in Kepler, we then had a better experience.

The custom workload is a Python-based application, therefore the Docker image is built on top of the Python base image. Once being built, the package list is updated in the container's environment from the configured sources, as well as download all dependencies needed for running the application. The container's environment now has the application script and will start by running the script. The image is ready for deployment to Kubernetes, and Kubernetes manages these pods, ensuring there are always 3 replicas running. The pod distribution to handle computation tasks is random, often 3 out of 5 worker nodes are in charged of executing.

The graph in Figure 2 is solely based on estimation, due to the precise value of carbon coefficient can not be determined. The configuration script has set the *Coal Coefficient* to 2.23, and *Natural Gas Coefficient* to 0.91.

The cluster took nearly 40 minutes to run the workload in the first time, and the execution time gradually decreasing in subsequent runs. The reason behind the shorter execution times can be caches, as caching helps storing frequently accessed data or computations in fast-access memory location, usually closer to the processor. Thanks to caching, the cluster can speed up the execution time.

6 Experiment Result

The successful deployment and configuration of Kepler on the cluster allow us to collect dataset of performance metrics. This section presents a detailed analysis of the experimental output obtained from Kepler.

Prometheus, in combination with Kepler in this experiment tool set, is designed to store data for a short amount of time, usually a few days before log rotation, which can lead to data lost. As extracted data is needed for analysis, it is crucial to mark timestamp whenever the experiment is executed. The result data can be huge, which possibly up to some gigabytes of metrics despite the execution time. Third-party tools for collecting data are unreliable, while snapshotting binary from Prometheus is incomprehensible. Additionally, the experiment machines had low-end hardware, which frequently led to memory shortages and crashes. The helper extracting script implemented by John Wickstrom has helped simplifying the extraction process, and formatting resulting data into JSON files with improved readability.

Besides the helper script, applying Kepler model server query function also help collecting data. By using Kepler's query function, only Kepler related data are scraped and store under one JSON output file. The *extract* method is specifically designed to retrieve data based on feature groups. The extracted data is then stored under CSV files, which offer several advantages for data management. CSV data files have

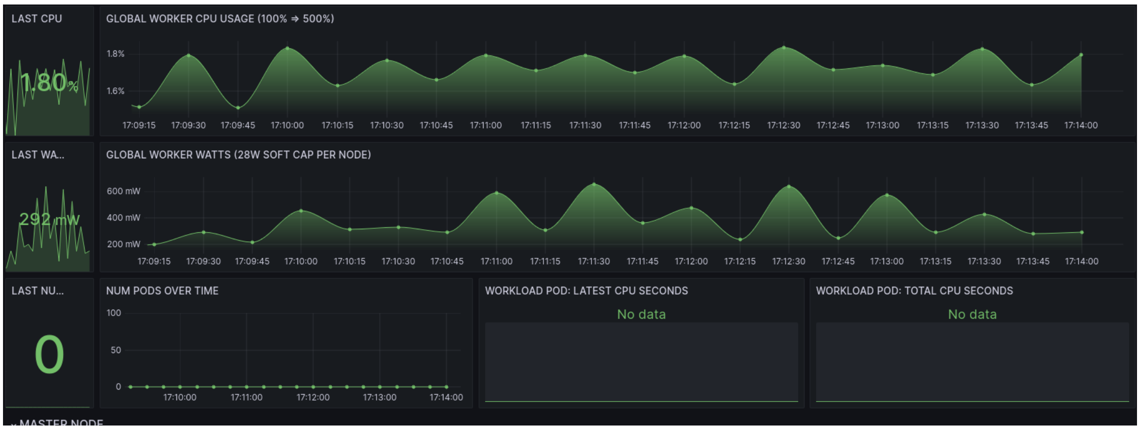

Figure 5: Visualization of the execution through Grafana.

Figure 6: Estimation of carbon emission.

straightforward structure, allowing easy analysis, reading and manipulating data. Moreover, CSV files are often the best choice for applying data to train ML models. Following the extrating step is the isolating step, which eliminate idle or background power from the power columns [3].

6.1 Overview of collected data

As mentioned above, the hardware configuration of this experiment does not support ACPI, therefore no metrics from this energy source is collected. While the absence of ACPI metrics may impact the overall accuracy of the training pipeline, RAPL metrics provide a sufficient basis for analysis within the scope of this experiment. However, despite the availability of RAPL metrics, some data points may still be missing or contain invalid values, which requires careful handling during the analysis. In total of 27 metrics, 16 of which are available with valid values, while 11 others metrics either contain values of 0 or invalid ones. Kepler states that when RAPL is unavailable, it can rely on its model server to estimate the metrics. Since RAPL is accessible in our hardware configuration, the model server is not utilized and therefore cannot provide estimates for any missing metrics.

kepler_node_core_joules_total
kepler_container_task_clock_ms_total
kepler_node_info
kepler_node_uncore_joules_total
kepler_container_cpu_instructions_total
kepler_container_core_joules_total
kepler_exporter_build_info
kepler_container_bpf_block_irq_total
kepler_container_uncore_joules_total
kepler_container_joules_total
kepler_container_package_joules_total
kepler_container_cache_miss_total
kepler_container_bpf_net_rx_irq_total
kepler_node_package_joules_total
kepler_container_bpf_cpu_time_ms_total
kepler_container_cpu_cycles_total

Table 1: Valid metrics

The commonality among the specified Kepler metrics (Table 1) lies in their focus on monitoring and analyzing resource utilization and performance in computing systems. These metrics collectively provide insights into various aspects of system operation, including energy consumption, CPU usage, task execution, and efficiency. The metrics also highlight what factors are critical and contribute to the overall performance of the cluster, for example, some CPU related metrics have significant greater values in those pods that were assigned with computing tasks compared to other idle pods.

Energy Monitoring: In computing, especially large-scale computing, energy monitoring plays an important role as it helps saving significant amount of resources. Metrics such as *kepler_container_core_joules_total*, *kepler_node_core_joules_total*, *kepler_container_joules_total*, and *kepler_node_package_joules_total* track energy consumption at different levels, such as CPU cores, sockets, and other host elements. These metrics reflect on the energy consumption of the components, for instance, the *kepler_container_core_joules_total*, which is an "aggregated package/socket energy consumption of CPU, DRAM, GPU, and other host components" metric in containerized environment. Due to the overhead on Prometheus, the metrics are often simplified to avoid this.
Another example is *kepler_node_core_joules_total*, which collects data about the energy that all CPU cores consumed in the socket, represent "the aggregation of all containers running on the node and operating system", and its values increased gradually over the course of running the experiment. CPU cores energy consumption not only contributing to the rise of total energy usage, but also generating heat. In larger scale computation environment, i.e. data centers, heat can be very problematic, as it requires the cooling system to cool down and therefore consume more power/energy. Moreover, energy monitoring helps to detect suspicious behaviors of servers, such as some CPU sockets may consume significant amount of energy while staying idle.

Performance Tracking: Metrics like *kepler_container_task_clock_ms_total*, *kepler_container_cpu_cycles_total*, and *kepler_container_bpf_cpu_time_ms_total* measure the time spent on tasks and the total CPU cycles used. Utilizing hardware counters is crucial in order to achieve precise examination of performance and resource efficiency. CPU time and CPU cycles are both used to assess CPU performance, while CPU time is a generic metric that record the total time to complete a task, CPU cycles provide a fine-grained measurement and handle the smallest unit of CPU work as well. In term of optimization, CPU cycle profiling provides insights into micro-architectural performance. This information is essential for evaluating application performance and identifying potential bottlenecks in processing. Another important metric for CPU utilization is *kepler_container_cpu_instructions_total*. CPU performance is one of the key factors causing cloud server capacity shortages [45]. With the growth of applied ML in resource utilization, CPU monitoring provides input for training and predicting resource usage, which is significant in effective planning to avoid unforeseen challenges. Users can allocate resources more efficiently, mitigate uncertainties and ensure that systems and operations run smoothly.

Resource Utilization: Metrics such as *kepler_container_cache_miss_total* and *kepler_container_bpf_net_rx_irq_total* provide insights into how well system resources, like memory and network interfaces, are being utilized. High cache miss rates can indicate inefficiencies that may require code optimization or hardware adjustments. This tracks the overall cache misses experienced by a specific container using hardware counters. Since there isn't a direct event counter for measuring memory accesses, last-level cache (LLC) misses serve as a good estimate for the number of memory accesses, as an LLC read miss usually leads to a corresponding read access to the main memory. The latter metric is for tracing "the total transmitted packets to network cards of the container using BPF tracing". Monitoring bandwidth usage, evaluating transmission delays, and detecting packet loss are crucial steps to ensure that applications avoid performance bottlenecks and continue to operate efficiently. Networking in cloud computing significantly impacts performance and power consumption. Poor network quality can lead to unnecessary rerouting, resulting in wasted energy [46].

Other metrics like *kepler_node_info*, *kepler_exporter_build_info* contain information (metadata) about the hardware architecture. Such information is static, therefore suitable for summarizing the configuration rather than data analysis. Overall, these metrics enable a holistic view of system performance, allowing developers and system administrators to make informed decisions regarding optimizations, troubleshooting, and resource management.

In term of unavailable metrics, feature group *cGroups* is the significant one. Control Groups (cGroups) are important in Linux-based environment, especially in context such as system administration, resource management, and containerization. With *cGroups*, we can have fine-grained resource allocation and monitoring such as CPU, memory, or network bandwidth, and isolation of resource usage among applications [5]. In Kepler eco-system, Kepler primarily collects total CPU time and total memory in bytes used by the container, total CPU time in kernel and in user space. The latter two metrics are for measuring time spent running code in kernel with tasks like system calls, and in user application including application processes, scripts respectively. Our current cluster setup has *cGroups* enabled and configured; however, the reasons for the missing *cGroups* metrics remain unexplainable. This is due to limitations in both human resources from our end and available documentations from Kepler. The absence of *cGroups* metrics does affect to the end result, as we lack the diversity of data recorded from different hardware counters. This also affects the training pipeline in terms of input data; however, in this thesis, we do not focus on the power model server.

Even though some important metrics are missing (0 values) such as container dram joules or BPF page cache hit, we have alternative metrics that can serve a similar purpose to some extent. The case is similar to *cgroups*, where we can identify the counters in our cluster and node configuration, but the tracing process is not functioning as expected. Therefore, we aim to figure out how these hardware counters collaborate with Kepler in future work.

6.2 Data analysis

Figure 4 illustrates the total cache misses collected during the experiment. The value of cache misses can reach up to hundred million, indicating that the CPU is handling such tasks that it can not find data it needs in the cache. During the initial experiment runs, the heavy computational tasks lead to slower processing time due to limited cache availability. The warm-up time is brief and as the cluster begins to handle the tasks, the trend in cache misses shifts noticeably. The variation in recorded data reflects the differing nature of each task, as the CPU processes them in unique ways.

In figure 5, we can observe a pattern in the experiment where, at the beginning,

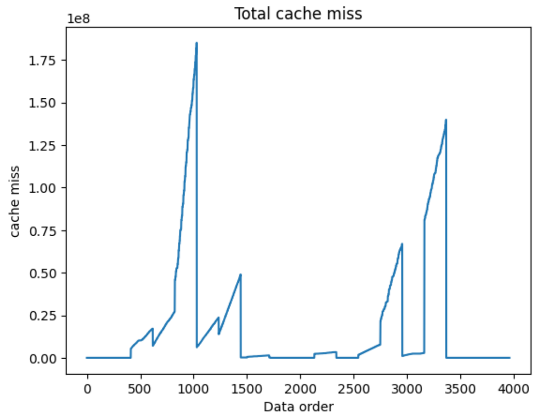

Figure 7: Total cache miss.

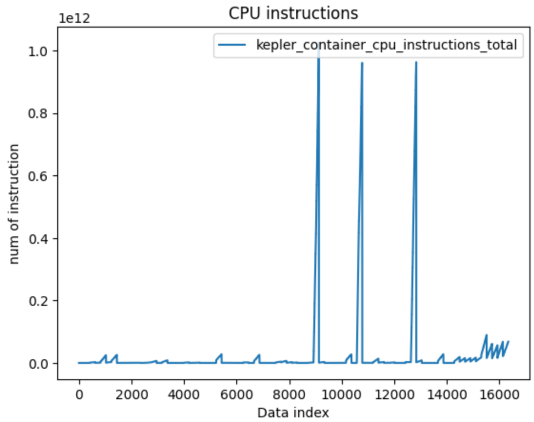

Figure 8: CPU total instruction.

there is minimal fluctuation in the metrics. However, starting around the 9000 mark, the values begin to skyrocket. Despite this increase, the values are not stable and exhibit significant variation over short durations. A few possible explanations for this include the fact that heavier tasks demand a greater amount of resources. In the custom workload script, tasks are organized from the simplest to the most complex, which accounts for the large spikes in CPU instructions around that mark.

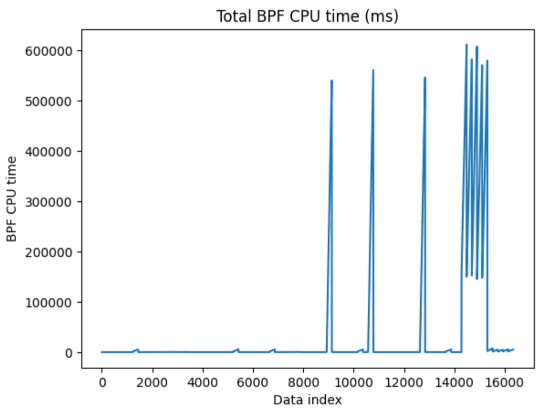

Figure 9: CPU time in ms traced by BPF.

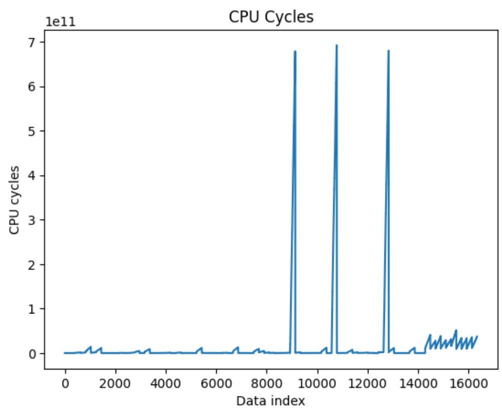

Figure 10: CPU Cycles

Figure 6 and 7 present CPU time spent and CPU cycles while running the experiment. The pattern is similar in the first half of the graphs, where values remain close to the baseline. Starting at 9000 mark we can see the values increased dramatically, both in time unit and cycle unit. In general, CPU cycle is the number of clock cycle the CPU uses to execute instructions and offers low-level view into the operation of the processor, while CPU time is the total time the CPU spends processing instructions for a program, a high-level view of performance. To some extents, CPU time provides a better practical insight for evaluating the performance, as it help optimize resource allocation and reduce energy consumption, which correlates significantly with cost. Amaral et al. [11] concluded based on their experiment that using software counter metric like CPU time have better result compared to exclusively using CPU cycles.

The difference between CPU time and CPU cycle data recorded at the same time is visibly from the graphs. While CPU cycle values drop considerably since 12000 mark with little fluctuations since 14000 mark, CPU time values show the opposite status. Between 14000 and 16000 mark, figure 6 presents multiple peaks. Based on the workload script, the last tasks focus on interactions with memory and cache. To observe cache hit, one function demonstrates a methodical approach to data access, where elements in a large array are incremented in a linear sequence. Randomness is also employed in the workload where the function access an arbitrary value and increase it. This randomness often results in the CPU frequently bypassing the cache, leading to increased cache misses. Consequently, the multiple memory accesses required by the random access pattern extend the overall CPU execution time.

Sustained high CPU load can result in increased power usage, as processors operate at higher frequencies and voltages to meet processing demands. This can lead to higher energy consumption and increased cooling requirements, ultimately contributing to elevated operational costs. Not only does it impact the overall costs but also contributes to increased carbon emissions. Particularly in data centers with substantial computational workload, either reducing CPU idle time or optimize execution time can help mitigate environmental impacts.

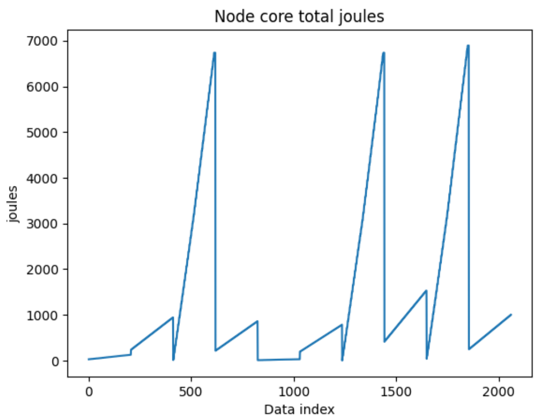

Figure 11: Node core total joules.

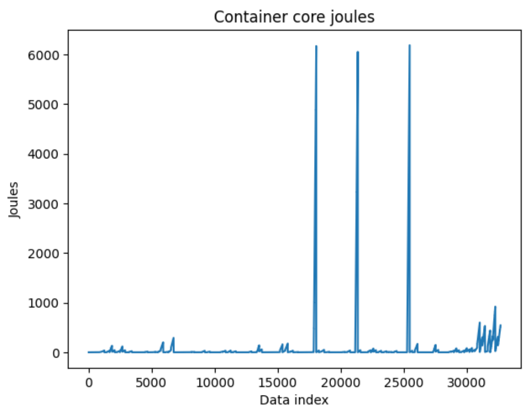

Figure 12: Container core total joules.

In figure 8 and 9, we observe the difference between node metrics and container metrics. While node metric gives a broad overview of energy usage, which is the total energy consumption of the CPU cores for the entire node in this case, container metrics provides more fine-grained information, allowing for detailed analysis and optimization of specific workloads. Node metric contains around 2000 values, while container metric can have up to more than 30000, a suitable number for using in training models. Joules values are important as they are input for capacity planning and cost management, helping anticipate future infrastructure needs based on energy consumption trends. For example, in enterprise data centers, engineers can route workloads to different nodes to maximize efficiency.

Kepler model server provides query function which produces the output data file and can be used for training models. From the result output file, we can extract and isolate by choosing feature group. Due to some missing metrics, the experimental data can not be utilized to train models as a compact input. Instead, we use extracted/isolated data to train some specific models targeting a certain feature group. Kepler offers several different models namely Linear Regression, Exponential Regression, or Polynomial Regression [3].

7 Discussion

7.1 Experimental Methodology

There are several studies and research aiming at simulating the cloud infrastructure, especially cloud storage service, and observing the energy consumption behavior. However, such studies demand advanced and remarkable resources to be as close to cloud infrastructure from data center as possible. The experiment of this thesis has smaller scope with local hardware setup, aiming to evaluate the performance of Kepler and analyze its collected data.

The overall experiment output is relatively small and are subjective to the local hardware setup. We observed several notable metrics that reflect performance; however, some were missed, as discussed in Section 6. Besides analyzing the values, we also consider the fact that a workload script written in Python can interact with the low-level components differently. Compared to C-a low-level, compiled programming language, Python trades off hardware-level control by abstracting away many of the technical details for user-friendly, simpler syntax. The execution of Python script relies on the interpreter like CPython, while C programs are compiled directly into machine code, which the CPU can execute natively. Therefore, in scenarios that workloads are written in compiled languages can reduce overhead of interpreters, and produce different outputs.

Tasks that involve heavy computation, such as numerical simulations or random memory accessing, often perform slower in Python. Exchanging for simplicity in designing workloads, we implemented the script in Python, which can lack of optimization for memory usage and CPU resources. However, the primary goal of the experiment is to evaluate Kepler performance and study the behavior of computing system by analyzing traced data, which can accommodate arbitrary types of workloads. By clarifying the differences between the programming languages, we aim to highlight that the collected data also contains error margins, and that the analysis is subject to interpretation and context.

7.2 Advantages and disadvantages of Kepler

Overview

Kepler was introduced in 2023 and is a CNCF sandbox project, aiming to assist the monitoring and optimizing power consumption efficiently in Kubernetes environment. By leveraging eBPF technology, Kepler probes energy-related system statistics and exports them as Prometheus metrics. It utilizes hardware and software counters to collect metrics, apply ML to build models with high accuracy in prediction. Amaral et al.[12] stated that Kepler is capable of delivering precise quantified energy consumption with "fine-grain granularity".

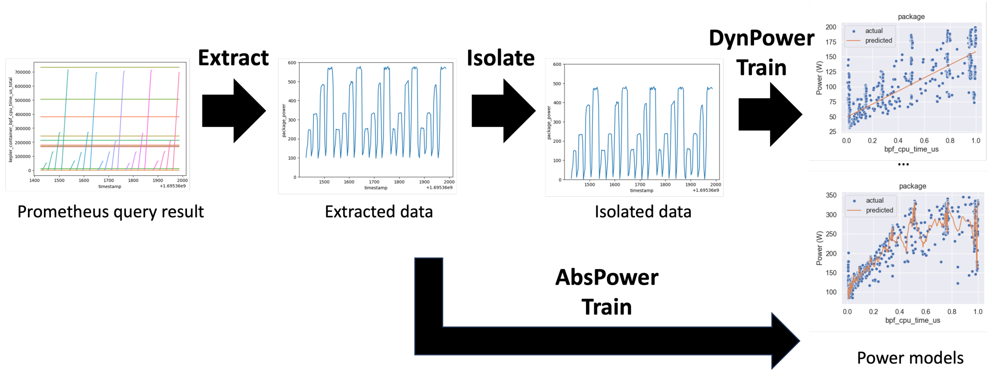

Figure 13: Kepler training pipeline [3].

The architecture of Kepler is described in detail in its documentation [3]. One important consideration when using Kepler to measure energy consumption is the distinction between Bare-metal and Virtual Machines (VMs) in public cloud environments. While Bare-metal nodes provide power metrics directly from their hardware components, VMs do not expose these metrics. Additionally, Kepler offers a Model Server for training power models, which can be deployed alongside Kepler. This helps Kepler choose the appropriate power model based on factors like CPU models and other requirements. The long-term goal is for Kepler to autonomously select the power model using the same logic as the Model Server.

The tool is not yet widely adopted in production and is still under development, with the latest version being 0.7.12 (updated 15th October, 2024). There are no reported adapters for use beyond research purposes, and most of the related research papers come from the tool's development team. Kepler is now a Cloud Native Computing Foundation (CNCF) sandbox project, under CNCF TAG Environmental Sustainability group, emphasizing the goal of advancing energy efficiency and sustainability in cloud-native environments [57]. Kepler Power models are introduced as an innovative step in energy monitoring, with various models available to provide the most accurate estimation while maintaining a low margin of error. Having applied Kepler in the experiment and thoroughly examined its capabilities, we will now discuss its advantages and disadvantages.

Advantages

In [12], the authors claimed that Kepler offers a fair allocation of system power consumption among applications and follows closely the Greenhouse Gas [32] protocol principles. Metrics from Kepler used for training models are real-time per-process metrics, which are different from common models that use aggregated data. There are no reported models that are employing the same methodology, which eventually distinguishes Kepler in the market. Kepler offers an extensive list of metrics from container level to platform level, help users to have a wider view on the behavior of the system.

Being a protocol tool, Kepler still have rooms for improvements. As an open-source tool, Kepler are free of charge, and the installation of Kepler is relatively simple and lightweight, encouraging developers to join and contribute to continuous improvement and feature additions.

Output data from *query* function is available in both JSON and CSV formats, in which CSV data is categorized automatically and filenames are metric names. The data files also include specific name of the pods, which is convenient for troubleshooting and monitoring.

Disadvantages

A considerably disadvantage of using Kepler is its limited documentation. The authors did a good job in explaining the fundamental concept and structure of Kepler, as well as Kepler model server. However, when it comes to practical usage, users would face problems that have not been discussed or addressed before. The development of Kepler now is generally based on the authors' infrastructure with some certain hardware models, and Kepler has not have the versatility and flexibility property yet to easily adapt to arbitrary setup.

Kepler features a complex architecture, and although the development team has created helper methods to automate and accelerate the process, a steep learning curve remains. It can take significant time and effort to fully adopt the technology. This thesis consists of a series of experiments. From configuring Kepler to adapt to the local workstation, which credit also goes to Roope Kettunen, to experiment with the model server. The compatibility issues between the tool and the workstation are time-consuming and produce performance overhead, which slows down operations and impacts application performance.

7.3 Connection with the carbon emission

The experiment data is solely an estimation from a protocol tool (Kepler) measuring a simple hardware setup, therefore the result is not compatible to compare with enterprise data centers. The insight of the experiment output serve as a conceptual ideas about the energy consumption behavior and how it contributes to the carbon emissions.

At its core, computing relies on the flow of electricity, which in most cases is generated from fossil fuels. The more energy a device consumes, the greater its greenhouse emissions. It is essential to consider hardware efficiency, software optimization; however, data such as CPU time and cycles do not directly reflect the carbon emission, but rather provide insights into computational workload. While direct and precise measurement of carbon emissions from specific computing tasks is challenging, estimation is a potential solution thanks to advanced software tools. From metrics like energy consumption, CPU resources statistic, we can assess the potential environmental impact of different computational processes. For instance, a high-performance computing task running on energy-efficient hardware and a low-carbon electricity grid will produce less carbon footprint.

By understanding the relationship between computational metrics and carbon emissions, developers can make informed decisions to reduce negative environmental impacts, optimize resources and achieve sustainability.

7.4 The sustainability challenge in cloud computing

The rapid expansion of cloud computing has undeniably transformed industries and daily life. However, this technological revolution comes with a significant environmental cost. Data centers, the backbone of cloud infrastructure, consume vast amounts of energy to power servers, storage systems, and cooling equipments. This energy consumption, often sourced from fossil fuel-based power plants, contributes substantially to greenhouse gas emissions. Furthermore, the complex interplay of hardware and software components within data centers makes it difficult to accurately measure and reduce their carbon footprint. While cloud providers have made significant strides in improving energy efficiency through techniques like virtualization, server consolidation, and advanced cooling systems, the increasing demand for cloud services continues to push the limits of sustainability.

To address these challenges, a multifaceted approach is necessary. Cloud providers must prioritize the adoption of renewable energy sources, such as solar and wind power, to reduce their reliance on fossil fuels. Additionally, investing in energy-efficient hardware and software, optimizing data center operations, and implementing sustainable cooling solutions are crucial steps. Users, too, can contribute to a more sustainable cloud ecosystem by making conscious choices. Selecting cloud providers with strong sustainability commitments, optimizing resource usage, and adopting energy-efficient practices can collectively reduce the environmental impact of cloud computing. By embracing sustainable practices and innovative technologies, the cloud industry can continue to drive innovation while minimizing its ecological footprint.

8 Conclusions

In this chapter, we summarize the findings and discuss the characteristics of our experiment result. We then conclude the thesis by outlining areas for future research.

8.1 Summary of findings

In this thesis, our objectives are study the energy consumption behavior and examine monitoring tool - Kepler, therefore analyze the correlation between energy consumption and the sustainability in cloud computing. We conduct an experiment by using Kepler and other energy monitoring helper tools to observe the behavior of the cluster. Thanks to John Wickstrom's local hardware station setup, we develop custom workloads where the interactions with low-level components and energy consumption can be observed with the monitoring tools. By applying containerized software like Docker, we successfully deployed the workloads in Kubernetes environment and detect the variations among the nodes while running the workloads.

We evaluate the result based on collected data, analyzing aspects such as CPU time, CPU cycles, estimated joules, and data from nodes that stay idle as the master node distributes the workloads arbitrarily among five worker nodes. The result data is collected with 3-second and 5-second intervals depending on the methods, offering detailed insights of the nodes. While the data are specific, there are also limitations from both sides - the monitoring software tools and our configuration, which led to the insufficiency of metrics. For instance, *cGroups* metrics group is critical in modern Linux systems and cloud environment, but we can not collect data from this feature group even though there is *cGroups v2* configured. The debug process was unsuccessful as there are very few sources from Kepler or the Internet that can help troubleshooting. Overall, Kepler performed fairly well in the experiment, providing better understanding of the system. The experiment presents a comprehensive overview of the advantages and disadvantages of a protocol-state tool like Kepler, and the performance of hardware systems.

The conducted experiment offers insights of the energy consumption patterns of the computing systems. By analyzing collected data, we gained valuable understanding into the factors influencing energy consumption, such as workload intensity, resource allocation strategies, and underlying hardware infrastructure. These findings suggest that integrating energy monitoring tools into computing systems is essential for improving both performance and energy efficiency through optimized resource management and efficient workload scheduling.

Additionally, this thesis presents a novel perspective of monitoring tool Kepler. Despite the sensationalism in cloud computing and energy sustainability, limited practical research has been done on applying tool like Kepler as this thesis. There are some research papers discussing the potential of Kepler and mostly from the development team. Evaluated based on its performance in this experiment and the available support documents, Kepler stands out as a potential monitoring tool thanks to its native characteristics, including the facts that Kepler is Kubernetes based and is a Prometheus exporter. However, while offering several beneficial methods for monitoring process, Kepler is lacking the generality of a product software tool that can easily adapt to various hardware device without significant difficulties. The development team is therefore working on to improve the tool and aiming to present the first official finalized version of the tool.

In conclusion, as cloud computing continues to evolve, it is important to prioritize energy efficiency to ensure a sustainable digital future. With the help of monitoring tool to understand the characteristic and behavior of cloud system, it is feasible to develop more eco-friendly solutions and practices, contributing to a greener planet.

8.2 Future work and improvements

As discussed throughout this thesis, the transparency in energy consumption and carbon emission of cloud services are crucial in maintaining the sustainability. Users with better insights of what they are using and paying for can make more informed choices that align with their needs and contribute to a reduced environmental footprint. Future research should focus on developing advanced cloud structure, and optimizing cooling systems in data centers. Moreover, investing in sophisticated energy-efficient algorithms and techniques for workload scheduling and resource allocation can enhance the performance. The integration of renewable energy sources, such as solar and wind power, into data center infrastructure can help minimize reliance on fossil fuels. By combining these strategies and fostering collaboration between research, industry, and governments, we can work towards a more sustainable future for cloud computing and data centers.

References

[1] *eBPF Documentation: What is eBPF?* Accessed: 2024-07-01. URL: https://ebpf.io/what-is-ebpf/.

[2] *Grafana documentation: About Grafana*. Accessed: 2024-06-01. URL: https://grafana.com/docs/grafana/latest/introduction/.

[3] *Kepler documentation: Kepler Metrics Overview*. Accessed: 2024-06-01. URL: https://sustainable-computing.io/design/metrics/.

[4] *Kubernetes Documentation*. Accessed: 2024-06-03. URL: https://kubernetes.io/docs/home/.

[5] *Linux Kernel Documentation: Cgroups*. Accessed: 2024-10-22. URL: https://www.kernel.org/doc/Documentation/cgroup-v2.txt.

[6] *Prometheus: Overview*. Accessed: 2024-06-01.

[7] *RedHat Documentation. Chapter 3: Understanding How SystemTap Works*. Accessed: 2024-10-03.

[8] *RedHat Documentation. Chapter 39: Getting started with SystemTap*. Accessed: 2024-10-03.

[9] Jorge Abril-Martorell, Luis del Barrio Castro, Manuel Sampedro, Juan Carlos Bernal, Matthew Humphreys, and Cristina Olivera Sánchez. Green data centers: Opportunities for decarbonization exploiting sustainable power-supply opportunities. Accessed: 2024-06-04.

[10] Ali Aghasi, Kamal Jamshidi, Ali Bohlooli, and Bahman Javadi. A decentralized adaptation of model-free Q-learning for thermal-aware energy-efficient virtual machine placement in cloud data centers. *Computer Networks*, 224:109624, 2023. DOI: 10.1016/j.comnet.2023.109624.

[11] Marcelo Amaral, Huamin Chen, Tatsuhiro Chiba, Rina Nakazawa, Sunyanan Choochotkaew, Eun Kyung Lee, and Tamar Eilam. Kepler: A framework to calculate the energy consumption of containerized applications. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*, pages 69–71. IEEE, 2023. DOI: 10.1109/CLOUD60044.2023.00017.

[12] Marcelo Amaral, Huamin Chen, Tatsuhiro Chiba, Rina Nakazawa, Sunyanan Choochotkaew, Eun Kyung Lee, and Tamar Eilam. Process-based efficient power level exporter. In *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, pages 456–467. IEEE, 2024. DOI: 10.1109/CLOUD62652.2024.00058.

[13] Manish Arora, Srilatha Manne, Yasuko Eckert, Indrani Paul, Nuwan Jayasena, and Dean Tullsen. A comparison of core power gating strategies implemented in modern hardware. *ACM SIGMETRICS Performance Evaluation Review*, 42(1):559–560, 2014. DOI: 10.1145/2591971.2592017.

[14] Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan. *The datacenter as a computer: Designing warehouse-scale machines*. Springer Nature, 2019. DOI: 10.1007/978-3-031-01761-2.

[15] Bryan Cantrill, Michael W Shapiro, Adam H Leventhal, et al. Dynamic instrumentation of production systems. In *USENIX Annual Technical Conference, General Track*, pages 15–28, 2004.

[16] Zhiwei Cao, Xin Zhou, Han Hu, Zhi Wang, and Yonggang Wen. Toward a systematic survey for carbon neutral data centers. *IEEE Communications Surveys & Tutorials*, 24:1–1, 01 2022. DOI: 10.1109/COMST.2022.3161275.

[17] Sunyanan Choochotkaew, Chen Wang, Huamin Chen, Tatsuhiro Chiba, Marcelo Amaral, Eun Kyung Lee, and Tamar Eilam. A robust power model training framework for cloud native runtime energy metric exporter. *arXiv preprint arXiv:2407.00878*, 2024. DOI: 10.48550/arXiv.2407.00878.

[18] Chaitali Choudhary, Inder Singh, and Manoj Kumar. A real-time fault tolerant and scalable recommender system design based on kafka. In *2022 IEEE 7th International conference for Convergence in Technology (I2CT)*, pages 1–6, 2022. DOI: 10.1109/I2CT54291.2022.9825441.

[19] Jakob Dybdal Christensen, Jens Therkelsen, Ivo Georgiev, and Henrik Sand. *Data centre opportunities in the Nordics: An analysis of the competitive advantages*. Nordic Council of Ministers, 2018.

[20] Peter Corcoran and Anders Andrae. Emerging trends in electricity consumption for consumer ict. *National University of Ireland, Galway, Connacht, Ireland, Tech. Rep*, 2013.

[21] David Couturier and Michel R Dagenais. LTTng clust: A system-wide unified cpu and gpu tracing tool for opencl applications. *Advances in Software Engineering*, 2015(1):940628, 2015. DOI: 10.1155/2015/940628.

[22] Sanjeev Das, Jan Werner, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. Sok: The challenges, pitfalls, and perils of using hardware performance counters for security. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 20–38. IEEE, 2019. DOI: 10.1109/SP.2019.00021.

[23] Howard David, Eugene Gorbatov, Ulf R. Hanebutte, Rahul Khanna, and Christian Le. Rapl: memory power estimation and capping. In *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '10, page 189–194, New York, NY, USA, 2010. Association for Computing Machinery. DOI: 10.1145/1840845.1840883.

[24] Victor Depoorter, Eduard Oró, and Jaume Salom. The location as an energy efficiency and renewable energy supply measure for data centres in europe. *Applied Energy*, 140:338–349, 2015. DOI: 10.1016/j.apenergy.2014.11.067.

[25] Victor Depoorter, Eduard Oró, and Jaume Salom. The location as an energy efficiency and renewable energy supply measure for data centres in europe. *Applied Energy*, 140:338–349, 2015. DOI: 10.1016/j.apenergy.2014.11.067.

[26] Mathieu Desnoyers and Michel Dagenais. Lttng: Tracing across execution layers, from the hypervisor to user-space. In *Linux symposium*, volume 101, 2008.

[27] Mathieu Desnoyers and Michel R Dagenais. The LTTng tracer: A low impact performance and behavior monitor for gnu/linux. In *OLS (Ottawa Linux Symposium)*, volume 2006, pages 209–224. Citeseer, 2006.

[28] Spencer Desrochers, Chad Paradis, and Vincent M. Weaver. A validation of dram rapl power measurements. MEMSYS '16, page 455–470, New York, NY, USA, 2016. Association for Computing Machinery. DOI: 10.1145/2989081.2989088.

[29] Hui Dou, Yong Qi, Wei Wei, and Houbing Song. Carbon-aware electricity cost minimization for sustainable data centers. *IEEE Transactions on Sustainable Computing*, 2(2):211–223, 2017. DOI: 10.1109/TSUSC.2017.2711925.

[30] Lide Duan, Dongyuan Zhan, and Justin Hohnerlein. Optimizing cloud data center energy efficiency via dynamic prediction of cpu idle intervals. In *2015 IEEE 8th International Conference on Cloud Computing*, pages 985–988. IEEE, 2015. DOI: 10.1109/CLOUD.2015.133.

[31] Saurabh Kumar Garg, Chee Shin Yeo, and Rajkumar Buyya. Green cloud framework for improving carbon efficiency of clouds. In *European Conference on Parallel Processing*, pages 491–502. Springer, 2011. DOI: 10.1007/978-3-642-23400-2_45.

[32] GHG. GHG: ICT sector guidance built on the ghg protocol product life cycle accounting and reporting standard, 2017. Accessed: 2024-10-18, URL: https://ghgprotocol.org/sites/default/files/2023-03/GHGP-ICTSG

[33] Laurence Goasduff. Gartner says cloud will be the centerpiece of new digital experiences, 2021.

[34] Make IT Green. Cloud computing and its contribution to climate change. *Greenpeace international*, 83, 2010.

[35] Brendan Gregg and Jim Mauro. *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X, and FreeBSD*. Prentice Hall Professional, 2011.

[36] Ali Hammadi and Lotfi Mhamdi. A survey on architectures and energy efficiency in data center networks. *Computer Communications*, 40:1–21, 2014. DOI: 10.1016/j.comcom.2013.11.005.

[37] Invest Iceland. Incentives for green investments, 2021. Accessed: 2024-11-10. URL: https://www.invest.is/incentives-and-support.

[38] Kashif Nizam Khan, Mikael Hirki, Tapio Niemi, Jukka K Nurminen, and Zhonghong Ou. Rapl in action: Experiences in using rapl for power measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 3(2):1–26, 2018. DOI: 10.1145/3177754.

[39] Kashif Nizam Khan, Zhonghong Ou, Mikael Hirki, Jukka K Nurminen, and Tapio Niemi. How much power does your server consume? estimating wall socket power using rapl measurements. *Computer Science-Research and Development*, 31:207–214, 2016. DOI: 10.1007/s00450-016-0325-4.

[40] Iman Kohyarnejadfard, Daniel Aloise, Michel R Dagenais, and Mahsa Shakeri. A framework for detecting system performance anomalies using tracing data analysis. *Entropy*, 23(8):1011, 2021. DOI: 10.3390/e23081011.

[41] Eun Kyung Lee, Indraneel Kulkarni, Dario Pompili, and Manish Parashar. Proactive thermal management in green datacenters. *The Journal of Supercomputing*, 60:165–195, 2012. DOI: 10.1007/s11227-010-0453-8.

[42] Diptyaroop Maji, Ben Pfaff, Vipin PR, Rajagopal Sreenivasan, Victor Firoiu, Sreeram Iyer, Colleen Josephson, Zhelong Pan, and Ramesh K Sitaraman. Bringing carbon awareness to multi-cloud application delivery. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems*, pages 1–6, 2023. DOI: 10.1145/3604930.3605711.

[43] Dan C Marinescu. *Cloud computing: Theory and Practice*. Morgan Kaufmann, 2022.

[44] Borja Martinez, Marius Monton, Ignasi Vilajosana, and Joan Daniel Prades. The power of models: Modeling power consumption for iot devices. *IEEE Sensors Journal*, 15(10):5777–5789, 2015. DOI: 10.1109/JSEN.2015.2445094.

[45] Karl Mason, Martin Duggan, Enda Barrett, Jim Duggan, and Enda Howley. Predicting host cpu utilization in the cloud using evolutionary neural networks. *Future Generation Computer Systems*, 86:162–173, 2018. DOI: 10.1016/j.future.2018.03.040.

[46] Toni Mastelic, Ariel Oleksiak, Holger Claussen, Ivona Brandic, Jean-Marc Pierson, and Athanasios V Vasilakos. Cloud computing: Survey on energy efficiency. *Acm computing surveys (csur)*, 47(2):1–36, 2014. DOI: 10.1145/2656204.

[47] Steven McCanne and Van Jacobson. The bsd packet filter: A new architecture for user-level packet capture. In *USENIX winter*, volume 46, pages 259–270. Citeseer, 1993.

[48] Christoph Möbius, Waltenegus Dargie, and Alexander Schill. Power consumption estimation models for processors, virtual machines, and servers. *IEEE Transactions on Parallel and Distributed Systems*, 25(6):1600–1614, 2013. DOI: 10.1109/TPDS.2013.183.

[49] Sangil Park and Jun-Ho Huh. A study on big data collecting and utilizing smart factory based grid networking big data using apache kafka. *IEEE Access*, 11:96131–96142, 2023. DOI: 10.1109/ACCESS.2023.3305586.

[50] Long Phan and Cheng-Xian Lin. A multi-zone building energy simulation of a data center model with hot and cold aisles. *Energy and Buildings*, 77:364–376, 2014. DOI: 10.1016/j.enbuild.2014.03.060.

[51] Vara Prasad, William Cohen, Frank Eigler, Martin Hunt, Jim Keniston, and J Chen. Locating system problems using dynamic instrumentation. In *2005 Ottawa Linux Symposium*, pages 49–64. New York, NY: IEEE, 2005.

[52] Ana Radovanović, Ross Koningstein, Ian Schneider, Bokan Chen, Alexandre Duarte, Binz Roy, Diyue Xiao, Maya Haridasan, Patrick Hung, Nick Care, et al. Carbon-aware computing for datacenters. *IEEE Transactions on Power Systems*, 38(2):1270–1280, 2022. DOI: 10.1109/TPWRS.2022.3173250.

[53] Neil Rasmussen. Calculating total cooling requirements for data centers. *White paper*, 25:1–8, 2007.

[54] Liz Rice. *Learning eBPF*. " O'Reilly Media, Inc.", 2023.

[55] Emanuele Rocca. Comparing systemtap and bpftrace, 2021. Accessed: 2024-10-27. URL: https://lwn.net/Articles/852112/.

[56] Dominik Scholz, Daniel Raumer, Paul Emmerich, Alexander Kurtz, Krzysztof Lesiak, and Georg Carle. Performance implications of packet filtering with linux ebpf. In *2018 30th International Teletraffic Congress (ITC 30)*, volume 1, pages 209–217. IEEE, 2018. DOI:10.1109/ITC30.2018.00039.

[57] Parul Singh and Huamin Chen. Introducing kepler: Efficient power monitoring for kubernetes, 2023. Accessed: 2024-10-17. URL: https://next.redhat.com/2023/08/22/introducing-kepler-efficient-power-monitoring-for-kubernetes/.

[58] Arsham Skrenes and Carey Williamson. Experimental calibration and validation of a speed scaling simulator. In *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 105–114, 2016. DOI: 10.1109/MASCOTS.2016.49.

[59] David Soldani, Petrit Nahi, Hami Bour, Saber Jafarizadeh, Mohammed F Soliman, Leonardo Di Giovanna, Francesco Monaco, Giuseppe Ognibene, and Fulvio Risso. ebpf: A new approach to cloud-native observability, networking and

security for current (5g) and future mobile networks (6g and beyond). *IEEE Access*, 11:57174–57202, 2023. DOI: 10.1109/ACCESS.2023.3281480.

[60] William Stallings. *Operating Systems: Internals and Design Principles, 9/e*. Pearson IT Certification, 2018.

[61] R Steinbrecher and Roger Schmidt. Data center environments. *ASHRAE J*, 53(12):e9, 2011.

[62] Mueen Uddin and Azizah Abdul Rahman. Energy efficiency and low carbon enabler green it framework for data centers considering green metrics. *Renewable and Sustainable Energy Reviews*, 16(6):4078–4094, 2012. DOI: 10.1016/j.rser.2012.03.014.

[63] Mikko Wahlroos, Matti Pärssinen, Samuli Rinne, Sanna Syri, and Jukka Manner. Future views on waste heat utilization–case of data centers in northern europe. *Renewable and Sustainable Energy Reviews*, 82:1749–1764, 2018. DOI: 10.1016/j.rser.2017.10.058.

[64] Wei Wang, Baochun Li, Ben Liang, and Jun Li. Multi-resource fair sharing for datacenter jobs with placement constraints. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1003–1014. IEEE, 2016. DOI: 10.1109/SC.2016.85.

[65] Vincent M Weaver, Dan Terpstra, and Shirley Moore. Non-determinism and overcount on modern hardware performance counter implementations. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 215–224. IEEE, 2013. DOI: 10.1109/ISPASS.2013.6557172.

[66] Qingxia Zhang, Zihao Meng, Xianwen Hong, Yuhao Zhan, Jia Liu, Jiabao Dong, Tian Bai, Junyu Niu, and M Jamal Deen. A survey on data center cooling systems: Technology, power consumption modeling and control strategy optimization. *Journal of Systems Architecture*, 119:102253, 2021. DOI: 10.1016/j.sysarc.2021.102253.

A Appendix: Customed Workload Script

```
import time
import numpy as np
import random

def cpu_intensive_task(duration):
    print("Starting CPU-intensive task...")
    end_time = time.time() + duration
    while time.time() < end_time:
        # Perform some CPU-intensive calculations
        [x**2 for x in range(10000)]
    print("CPU-intensive task completed.")

def memory_allocation_task(size_mb, duration):
    print("Starting memory allocation task...")
    end_time = time.time() + duration
    arrays = []
    while time.time() < end_time:
        # Allocate large arrays to simulate memory usage
        array = np.zeros((size_mb * 1024 * 1024 // 8,), dtype='float64')
        arrays.append(array)
        time.sleep(0.1)  # Adjust the sleep time to control allocation rate
    print("Memory allocation task completed.")

def memory_access_pattern_task(size_mb, duration, pattern='sequential'):
    print(f"Starting memory access pattern task ({pattern})...")
    array = np.zeros((size_mb * 1024 * 1024 // 8,), dtype='float64')
    end_time = time.time() + duration
    if pattern == 'sequential':
        while time.time() < end_time:
            for i in range(len(array)):
                array[i] = random.random()
    elif pattern == 'random':
        while time.time() < end_time:
            for _ in range(len(array)):
                array[random.randint(0, len(array) - 1)] = random.random()
```

```python
    print(f"Memory access pattern task ({pattern}) completed.")

def matrix_computation():
    size = 10000
    matrix_a = np.random.rand(size, size)
    matrix_b = np.random.rand(size, size)

    # Perform a matrix multiplication
    result = np.dot(matrix_a, matrix_b)

    return result

def sequential_access():
    # Access data sequentially to increase cache hits
    array_size = 10**6
    data = np.zeros(array_size)

    for i in range(array_size):
        data[i] += 1

def random_access():
    # Access data randomly, which may lead to more cache misses
    array_size = 10**6
    data = np.zeros(array_size)

    for _ in range(array_size):
        i = np.random.randint(0, array_size)
        data[i] += 1

def main():
    cpu_duration = 150      # Duration of CPU-intensive task in seconds
    mem_size_mb = 100       # Size of memory to allocate/access in MB
    mem_duration = 150      # Duration of memory tasks in seconds

    # Perform CPU-intensive task
    cpu_intensive_task(cpu_duration)

    # Perform memory allocation task
    memory_allocation_task(mem_size_mb, mem_duration)

```

```
    # Perform memory access pattern task (sequential)
    memory_access_pattern_task(mem_size_mb, mem_duration, pattern='sequential')

    # Perform memory access pattern task (random)
    memory_access_pattern_task(mem_size_mb, mem_duration, pattern='random')
    print("Starting heavy computation...")
    matrix_computation()
    print("Heavy computation done.")

    # Sequential access to data
    print("Starting sequential data access...")
    sequential_access()
    print("Sequential access done.")

    # Random access to data
    print("Starting random data access...")
    random_access()
    print("Random access done.")

if __name__ == "__main__":
    main()
```