Vrije Universiteit Amsterdam

Master Thesis

BenchFrame: A Framework for Benchmarking Power Monitoring Tools

Author: Thimo Wuttge (2829081)

1st supervisor:	Tiziano De Matteis
daily supervisor:	Matthijs Jansen
2nd reader:	Daniele Bonetta

A thesis submitted in fulfillment of the requirements for the VU Master of Science degree in Computer Science

August 18, 2025

Abstract

The energy consumption of data centers has become an increasingly important issue in recent years, driven by the growth of cloud computing, global connectivity, machine learning, and other data-intensive applications. To keep data and services available around the clock, data centers operate continuously – this can introduce significant energy consumption. Monitoring and optimizing their energy usage is essential not only for reducing operational costs but also for addressing environmental concerns, as energy is not always generated in a sustainable or resource-efficient manner.

This research project contributes to improving energy efficiency in data centers by investigating methods to monitor the power consumption of servers. A single data center may house thousands of servers, each contributing to the total energy footprint. As such, optimization efforts must begin at the server level.

To address this concern, we design and implement `BenchFrame`, a power benchmarking framework. We survey different power monitoring techniques and benchmarking approaches during the design phase. Afterwards, we realize the design by implementing `BenchFrame`. We assess the selection of power monitoring tools by executing various benchmark experiments on the framework and conduct an in-depth analysis on the monitoring tools.

Through a comprehensive analysis of the collected data, we find that software tools typically offer more fine-grained monitoring capabilities, whereas hardware tools provide more stable and consistent readings. Additionally, internal tools often measure only a subset of system components, limiting their ability to capture total system energy consumption. This allows an analysis of system components, but limits the analysis of the whole system.

Contents

Acronyms

ACPI Advanced Configuration and Power Interface. 18

BMC Baseboard Management Controller. 17, 26, 27

IPMI Intelligent Platform Management Interface. 16, 17, 26

NIC Network Interface Controller. 33, 36

PCH Platform Controller Hub. 17

PDU Power Distribution Unit. 9, 17, 18, 27, 28, 37

PSU Power Supply Unit. 9, 17

RAPL Intel® Running Average Power Limit. 16, 18, 24, 25, 41, 44, 45, 47–51, 53, 54, 63, 64, 71

SUT System Under Test. 37

UPS Uninterruptible Power System. 9

1 Introduction

Throughout the last decades, the importance of computers in private as well as professional work environments grew to a point where almost 70 % of the world's population use the internet through a device at the start of 2025.(1) This transformation in human behavior was made possible by inventing powerful technologies. One of these technologies, that grew in particular popularity throughout the last years, is the cloud. As of 2023 95% of companies in Europe stated, that they use cloud resources in their everyday business.(2) Hereby the range of available functionalities provided by the cloud ranges from storing data in cloud storage like Google Drive, iCloud, etc., to running complete IT infrastructures in a cloud environment. The cloud services require high computation and storage resources, which are provided by multiple connected data centers. The servers in these data centers do not only require scarce materials but also run with high electricity demands. Therefore, several energy saving techniques try to optimize the efficiency of the servers to contribute to a lower energy consumption of the whole data center.

The high energy consumption does not only result in additional costs for the provider but also harm the environment.(3) Therefore, energy saving techniques contribute to a better monetary as well as environmental outcome for the providers. Additionally, more and more regulations are being established that require the data center maintainers to provide energy usage reports and limit their energy consumption to meet certain legal restrictions.(4) Therefore, a comprehensive energy consumption analysis is not only beneficial for the maintainers but necessary to ensure a sustainable and environment-friendly future for all beings. To analyze different energy saving strategies, the energy consumption of the servers must be monitored and evaluated. For this purpose, power measuring tools can be integrated. The providers can choose from various options depending on factors like accuracy, complexity, and many more. The list of available options is long and every datacenter maintainer has different requirements. Integrating power monitoring tools in a data center can, therefore, become a complex task, where the selection of the meter itself represents a fundamental challenge.

To understand the advantages and disadvantages of different power monitoring tools, they must be analyzed under various workloads. Each tool may respond differently and can be more sensitive to varying tasks. The workloads a server has to process may stress different parts of the system. Every server component contributes to the total energy consumption in a certain degree. Therefore, it is not only necessary to monitor the total power draw, but additionally, to analyze the amount of energy a component-specific workload requires. This approach enables a comprehensive picture of the system's energy behavior.

1.1 Problem Statements

To optimize the energy consumption of a data center, the power behavior of servers must be monitored and evaluated. Numerous approaches allow the server maintainer to monitor and analyze various metrics. Depending on their specification, the provided metrics can vary in multiple factors like granularity, accuracy, or many more. Each system comes with preinstalled tools, but different vendors also provide various extensions that can be included in the system. This vast landscape of monitoring tools can make the decision of the most suitable tool complex.

Therefore, analyzing power monitoring tools and the energy consumption of servers is a complex task. For this purpose, we establish a power benchmarking framework, which provides insights in the energy consumption of servers and different motioning techniques. By executing benchmark experiments, we can evaluate each power monitoring tool under various conditions and investigate the power behavior of the server and how the tools monitor it. This benchmark-based evaluation allows us to draw conclusions about different aspects of the tools. These conclusion allow server maintainers and researchers to better understand how current power monitoring tools work and which benefits they provide.

To design and engineer a power benchmarking framework, we need to address three main problems. First, the current landscape of tools must be assessed. This assessment is based on information provided by the vendors or other researchers and does not include an experimental investigation. A second investigation is necessary to gain an understanding how sensitive the tools are to various conditions and workloads. To execute this second assessment, we utilize micro-benchmark tools. Therefore, we need to establish an understanding of how benchmark tools work, which tools are necessary to provide a comprehensive system analysis, and how we can combine the benchmark tools, the monitoring tools, and the tools assessment in one process. After the theoretical analysis of the monitoring tools and the planing, design and implementation of a benchmarking framework, the experimental investigation of the tools can take place. Hereby, the monitoring results need to be analyzed and compared. The main challenge is to establish and design a meaningful evaluation process.

P-1 *How can we evaluate the energy consumption of servers?*
To optimize the energy consumption of data centers, first, the power of the servers must be monitored. This is a fundamental step in understanding the behavior of the servers under different workloads. The range of tools for this purpose is vast and understanding the differences of the tools is an important first step. The tools can vary significantly in regard of multiple factors like accuracy or data granularity. Each tool also provides different interfaces to access their results. Therefore, to choose the correct power monitoring tools, available options must be assessed and the most suitable option selected.

P-2 *How should a monitoring framework be designed and implemented?*
After a set of tools is selected, their behavior and accuracy must be assessed. For this purpose, we use benchmarking tools. These tools enable the user to stress a specific component of the server by executing a task extensively. Using this approach, common server workloads can be simulated. The range of benchmarking tools is nearly endless with various use-cases for each single tool. To decide on a applicable set of benchmarking tools, user requirements must be analyzed and a decision on the most appropriate tools taken. After a set of tools is selected, the correct orchestration of benchmark testing and power measuring need to be executed. Hereby certain design requirements must be met and a correct execution established.

P-3 *How can the different power monitoring techniques be assessed and analyzed?*
After a selection of power monitoring tools and benchmark test is taken, and a framework to orchestrate the correct execution is established, the experiments can be carried out. Subsequently, the collected data must be analyzed and evaluated. Hereby observations about the power monitoring tools and the server itself should provider deeper insights into the behavior of the energy consumption under different conditions. To achieve this, an exploratory analysis of this data is required and the correct statistical analysis be applied.

1.2 Research Questions

Based on the problem statements we formulate three research questions. These research questions address the fundamental challenges of this research project. These are the correct selection of power monitoring tools, how we tested their behavior using benchmark tests, and how we evaluated the results.

RQ-1 *Which techniques and tools are available to monitor server energy consumption and how do they differ?*

To analyze the energy consumption of servers, numerous tools can be utilized. These tools can vary based on multiple characteristics and factors. The first step in investigating these tools is to find categories to sort these tools and highlight differences of each category:

RQ-1.1 *What tools exist, how can they be classified, and what are their characteristics?*

Furthermore, we want to analyze how we can use these tools and what kind of information is provided. For this purpose, we will look at the scope and the monitoring results of these categories in more detail. This will help us during the tool decision process and the analysis of the experiment results:

RQ-1.2 *How can we utilize the monitoring tools, how are the monitoring results provided, and how can we extract them?*

After the power monitoring tools are assessed, a final selection of tools must be decided on. For this purpose, various requirement need to be established so that the selection satisfy them:

RQ-1.3 *How can we select a comprehensive set of tools and which requirements should they satisfy?*

RQ-2 *How can power monitoring tools be evaluated?*

To get a better understanding of power monitoring tools, we want to assess them under various conditions. To this end, different components of the server should be stressed with common tasks. For this purpose, we first need to evaluate which categories of server tasks exist and what components are relevant for these tasks:

RQ-2.1 *Which tasks must a server process and which components are involved?*

After a categorization is established, a selection of benchmarking tools for each purpose should be established. Hereby, various selection requirements must be chosen and we need to ensure they are satisfied by the selected benchmarking tools:

RQ-2.2 *Which selection requirements should be used and which tool selection satisfies these?*

To finally assess the power monitoring tools and the energy consumption of the server, we need to design and implement a framework which orchestrates the correct execution of benchmarking, monitoring, and data processing. For this purpose we need to define design requirements and create a implementation that satisfies these requirements:

RQ-2.3 *Which requirement need to be met by a benchmarking framework?*

RQ-3 *How do different workloads influence the overall system's energy consumption, and to what extent can power monitoring tools capture this behavior?*

The final step to analyze the energy consumption of power monitoring tools is to execute the experiments and evaluate the resulting data sets. First, we need to establish a concrete plan on how we want to execute the experiments, what our hypotheses are, and how we address these in our experiments:

RQ-3.1 *How can we design, execute and interpret the benchmark experiments?*

Subsequently, the experiments can be executed and the results analyzed. The experiments should provide deeper insights into two research gaps. We want to analyze and address the behavior of the server itself. For this purpose, we address wether the energy consumption of the server behave according to the task executed:

RQ-3.2 *How does the server's energy consumption change under various tasks?*

Additionally, we want to use these results to verify wether the power monitoring tools analyzed the energy consumption in the correct way. To do so, we compare the result from the tools and evaluate their monitoring results based on various criteria:

RQ-3.3 *How can the power monitoring tools be evaluated based on the benchmarking results?*

1.3 Research Methodology

To answer the research question, we apply various techniques. These range from reference-based research to practical experiments:

M-1 Through a **quantitative survey of current research advances**, we are able to assess the state of current research in a specific domain. We use this methodology to evaluate technologies and tools, and decide on a selection that fits our purpose. To achieve this, we first specify distinctive categories that the technologies fall into. Afterwards, we can establish requirements which need to be satisfied by the technology in question. Subsequently, we can make a comprehensive selection of technologies for the purpose of our research question. This approach allows us to decide on a selection of technologies and tools relevant for power monitoring as well as benchmark testing.

M-2 We use an **workflow design** approach to include various technologies in our benchmarking framework. We use a requirement-based design approach inspired by Pohl et al.(5). We first establish functional and quality requirements and ensure that our design decisions align with these requirements. Hereby, the workflow execution always follows the same sequence:

Begin monitoring – Run experiment – Stop monitoring and evaluate results

Additionally, we include a **plug-in strategy** that allows us to easily add and remove technologies and tools. This design approach allows us to assess multiple technologies and evaluate the system under various conditions. For the purpose of this design strategy, we first establish requirements and then design a suitable system.

M-3 With the help of **benchmarks experiments** we realize our research design. Hereby, we include two different kind of benchmark experiments:

- Through component-workload benchmarks we can evaluate the influence of workloads focused to stress a certain server component on the energy consumption and how this behavior is recorded by the monitoring tools.
- Behavioral benchmarks allow us to further investigate the functionality of tools.

This allows us to analyze and proof theory-based assumptions. These benchmarks allow us to address research questions and compare various technologies under the same system conditions. For the selection of the benchmark test we analyze the different components we want to evaluate. Afterwards, we select various tools to execute the tests and stress the system components. This selection is based on requirements which we establish beforehand.

1.4 Thesis Contributions

C-1 *Classification and assessment of current power monitoring tools*
Based on a research review, we provide an overview of various power monitoring

technologies and approaches how the energy consumption of servers can be monitored and evaluated. We explore different options how the energy consumption of server can be monitored and establish a classification of these options. By doing so, we highlight various advantages and restrictions these technologies. To further investigate the different ways to monitor the energy consumption of servers, we choose a representative tools to cover each option in our classification and further inspect it through benchmark experiments.

C-2 *A benchmarking framework that enables reproducible power monitoring experiments*
Through the analysis of various benchmarking tools and the design of a benchmarking and implementation of a benchmarking framework, we provide an instrument for testing power monitoring tools. This framework orchestrates the execution of benchmark tests while measuring numerous metrics via power monitoring tools. The framework is used to inspect different power monitoring tools under various conditions and analyze their behavior.

C-3 *Evaluation of the energy consumption of servers and assessment of power monitoring tools based on various criteria*
With the evaluation of the data provided by the different benchmark experiments, we provide insights into the behavior of a server's energy consumption under various conditions. Through these experiments we can analyze the influence of different component-specific workloads on the system. Hereby we focus on the CPU, memory, storage, and networking. Additionally, we evaluate the power monitoring tools by comparing their results and assessing them based on various criteria. In this analysis we investigate the different behavior of the tools, and highlight differences in granularity, stability and latency of the different tools.

1.5 Plagiarism Declaration

I confirm that this thesis work is my own work, is not copied from any other source (person, Internet, or machine), and has not been submitted elsewhere for assessment.

To understand more about plagiarism policy at VU Amsterdam, see https://vu.nl/en/about-vu/more-about/academic-integrity.

1.6 Thesis Structure

The rest of this thesis is structure as follows: Section 2 provides additional information about the background of power management in data centers. In Section 3, we discuss the design requirements and the resulting design decisions of the power-benchmarking framework. In Section 4, we introduce the implementation of `BenchFrame`, the power benchmarking framework and how the benchmark experiments are executed. In Section 5, we present how we designed the benchmark experiments. Section 6 presents the results and evaluates them. We provide observations taken from these results. In Section 7, we compare our experiment and the results to related research. In Section 8, we summarize our findings in the Conclusion.

2

Background

Data centers can be considered computational warehouses, which include all necessary resources for cloud services to run. They consist of many server racks, each of which holds multiple compute nodes. To power this whole infrastructure, data centers require complex power infrastructure.

2.1 External Power Management

A fundamental control mechanism is the Uninterruptible Power System (UPS). This system enables a constant power supply for the data center by performing three main tasks. First, it switches between different power sources to always provide the most suitable power, depending on demand and supply shortages. Second, it handles different forms of energy storage, which can be used during low-supply phases. Third, it evens out the incoming energy to avoid voltage spikes or sags.

The next system in the power management of the data center is the set of Power Distribution Units (PDUs). These units distribute power between the different racks and provide constant power to each node. We can consider everything up to this point as external power management. From here onward, the compute node manages the power distribution to its internal components.(6)

2.2 Internal Power Management

Internal power management begins as soon as the power enters the compute node. Each node is equipped with at least one Power Supply Unit (PSU). This component distributes power between the different components in the node. Components can be divided into two categories: *Non-IT power* encompasses all components that do not directly provide computational resources for the system. Classic examples of this are cooling systems or LEDs. *IT power*, on the other hand, includes all compute resources such as the motherboard, storage components, or networking devices. Both categories influence the energy consumption of the compute node to a certain degree. Additionally, it is only possible to measure the direct relationship between *IT power* components and energy consumption, as the state and utilization of *non-IT power* components are often not provided.(7)

3

Design of a Power-Benchmarking Framework

To monitor and evaluate the energy consumption of a server, we design and implement a benchmarking framework. The first part of the design focuses on the high level structure of the framework. This includes planning the orchestration of benchmark task executions, monitoring, as well as handling the monitoring results. Afterwards, we select a set of monitoring tools by establishing a categorization of monitoring techniques and ensuring that each category is covered by at least one tool. By including multiple monitoring tools, we can cross-validate measurements, compare tool-specific characteristics such as accuracy, granularity, and stability, and ensure broader coverage of system components and measurement methodologies. In the final design step, we address the benchmarking tools. In this step, the tool requirements, the tool selection, and the design of the benchmark tasks takes place. In the last step we design the benchmarking framework.

3.1 Power-Benchmarking Framework

The first design step for the benchmarking framework is to establish requirements for the framework architecture. First, we establish the design requirements and, then, present how we integrate these requirements in our system design.

3.1.1 Design Requirements

To establish requirements for the benchmarking framework, we focus on technical limitations, as well as user restrictions, which must be fulfilled by the final design. Therefore, we follow the design approach of Pohl et al.(5) and divide the list of requirements into **functional** and **quality** requirements:

F-Req-1: **Portability**
To make the benchmark experiments executable on multiple systems, the framework needs to be independent of any system-specific configurations.

F-Req-2: **Extensibility**
The framework should be easily extensible, so that new benchmark tests can be added or adjusted.

F-Req-3: **Integrity**
To ensure that the monitored and recorded power data is not adjusted or tampered, we need the framework to store the raw data without any processing steps.

Q-Req-1: **Usability**
The framework should be easy to install and ready to use without extensive setup.

Q-Req-2: **Consistency**
The framework should behave consistently and perform identically across repeated executions.

3.1.2 Design Concept

An overview of the final framework design can be seen in Figure 3.1. The execution of the framework is split up into three distinct processes. The main **orchestration process** orchestrates the execution of the total framework. First, it loads system specific configurations ①. This step is necessary to satisfy **F-Req-1**. Afterwards, a benchmark test is loaded from the `benchmark` repository ②. This repository holds all benchmark tests and can easily be extended to align with **F-Req-2**. Next, the monitoring process can be started ③. This process reads the data collected by the different monitoring tools ④ and writes the raw values to a `results` repository ⑤. This specification satisfies **F-Req-3**. After this process is started, the main **orchestration process** executes the loaded benchmark test in a decoupled third process and waits ⑥. After the benchmark process finishes, the **measurement process** is stopped. At this point, one benchmark test iteration is finished

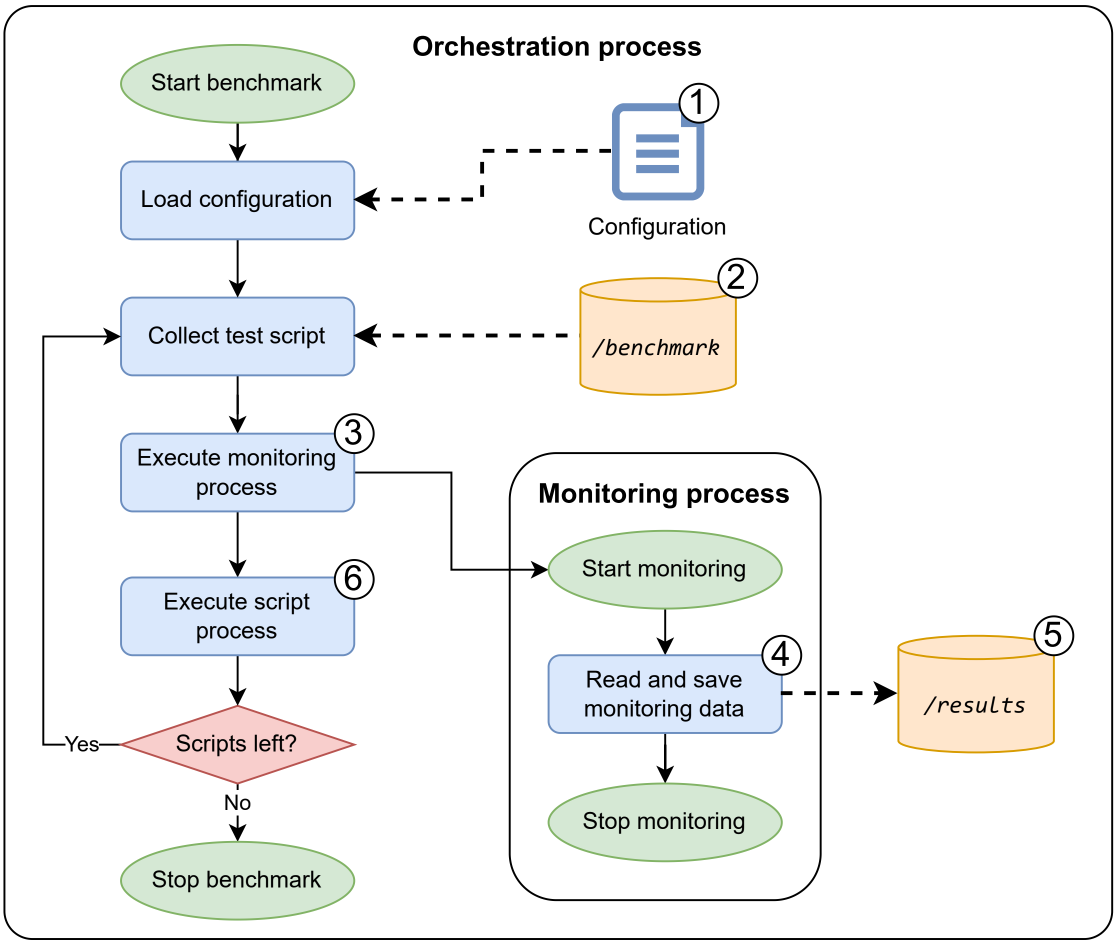


Figure 3.1: Control flow of the benchmarking framework.

and the benchmark framework either continues by loading the next benchmark test or finishes the execution.

To ensure the usability requirements are met, only the **orchestration process** needs to be started. This process then oversees the correct execution of the framework. Therefore, only the necessary power monitoring tools need to be configured. Furthermore, by dividing the framework into three processes, we can ensure that each one of these runs on their own and does not interfere with the other. This allows the framework a consistent execution over multiple runs.

3.2 Monitoring Metrics

Various power monitoring tools are provided by vendors, component producers and researchers. To analyze differences in these tools, we establish a categorization and select a representative tool to cover each option in the categories.

We base our categories on a taxonomy established by Lin et al.(8). They distinguish four different monitoring approaches, which are:

- Instruments-based tools
- System-based tools
- Software tools
- Simulations

As this research projects focuses on implemented solutions, we exclude simulations in our tool assessment. Therefore, we focus on the first three tool categories.

To identify differences of these categories, we distinguish three tool characteristics, as presented in Figure 3.2. These are the *implementation method*, the *monitoring scope*, and the *integration.* While there can be multiple other properties and features in which the tools may differ, we focus on fundamental attributes which influence the quality and scope of the power readings.

The implementation method can include software-based and hardware-based tools. Software-based tools comprise all tools which only use software-based models to estimate the power consumption, whereas hardware-based tools are only using hardware meters. The second distinction we make is the scope of the tools. We differentiate between system- and component-scoped monitoring. Component-scoped monitoring tools only monitor the energy consumption of certain components of the system, like the CPU or the RAM. This distinction is especially relevant during the evaluation of the monitoring, since component-scoped tools only include monitor a fraction of the total system. Lastly, the tools can be integrated in different ways into the system. Hereby, we distinguish between internal and external tools. External tools are not part of the system and extract information from the system without interfering with its execution. Internal tools may also be additionally added but are part of the system.

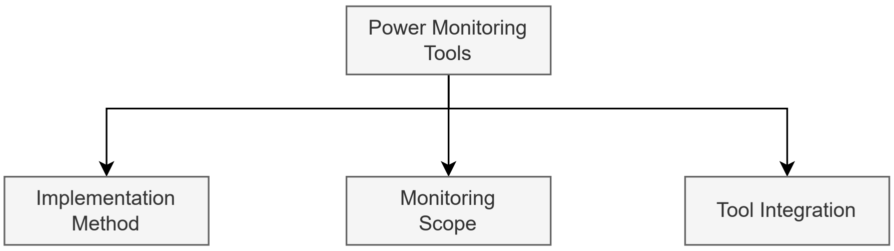


Figure 3.2: Categorization of different monitoring tools.

3.2.1 Requirements

To establish a selection of relevant power monitoring tools, we decided on several requirements based on technical feasibility, research relevance, and reproducibility. These requirements were derived from the research questions of this study — namely, to analyze tools and their behavior under different workloads, and to create a benchmarking framework that can be reused in other research settings.

These requirements are:

Req-1: **Coverage**

To assess the different approaches of each category, we establish a set of power measurement tools. Hereby, the selection of tools need to cover all options of the categories presented in Section 3.2. This requirement allows us to make observations about specific power monitoring tool categories. Therefore, further research project can use our insights to draw conclusions about other tools belonging to a specific tool category.

Req-2: **Data origin**

We require the tools to be the origin monitoring source. By doing so, we avoid added abstraction layers, which allows us to better understand the origin of the monitored data. Furthermore, this enables us to make the outcome of this research project more transparent, so that other researchers can build on top of our results.

Req-3: **Setup**

We exclude tools with extensive hardware or software setup to support reproducibility of our experiments. Hereby, other research projects can easily execute our benchmarking framework on their systems, without extensive setup beforehand. By including this requirement, we can ensure reproducibility of our exper-

Name	**Implementation**	**Scope**	**Integration**
Intel® RAPL	*Software*	*CPU and RAM*	*Internal*
IPMI/Redfish	*Hardware*	*System*	*Internal*
Netio PowerPDU 4KS	*Hardware*	*System*	*External*

Table 3.1: Overview of employed power monitoring tools.

iment and it enables other researchers to run the framework with ease.

Req-4: **Interface**

The tool must provide a command-line interface to allow seamless integration into the benchmarking framework. This requirement ensures that the tool can be programmatically controlled, enabling automated execution and consistent measurement runs without manual intervention. Since the benchmarking framework is designed to orchestrate experiments end-to-end, tools that rely solely on graphical user interfaces or manual input would disrupt reproducibility and introduce unnecessary overhead.

3.2.2 Tool Selection

With these requirements, we decide on three monitoring tools to include in `BenchFrame`. These are Intel® Running Average Power Limit (RAPL), Intelligent Platform Management Interface (IPMI)/Redfish, and Netio 4KS. An overview of the selected monitoring tools can be seen in Table 3.1. Hereby, one can already see that this selection satisfies **Req-1**, as we have a representative tool for each category option.

RAPL:

RAPL is a software-based power monitoring tool implemented and maintained by Intel®. It is included in all systems running the *x86* architecture or the extension *x86-64*. RAPL provides power metrics per CPU and its dedicated RAM. As it is included via the processor architecture, it is a native monitoring tool and, therefore, satisfies **Req-2**. Additionally, it is integrated by default and needs no setting up, which satisfies **Req-3**. **Req-4** is achieved on Linux through the *Power Capping* framework(9), and on Windows and macOS via the *Intel® Power Gadget API.* (10)

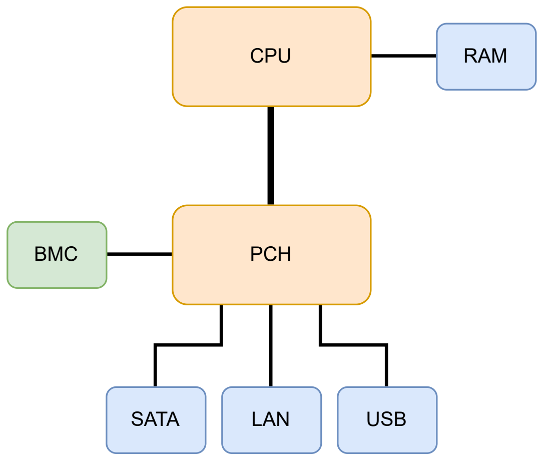


Figure 3.3: System block diagram explaining the BMC.

IPMI/Redfish:

The second tool we include in our analysis is IPMI, as well as it's newer implementation Redfish. These tool represent the interface to the BMC. The BMC is an additional hardware component which is installed on the motherboard of the system. Their purpose is to allow maintenance without accessing the server itself. Additionally, the user can read various information from this chip. We utilize this technology to read power usage data.(11) The BMC is connect to the Platform Controller Hub (PCH) and reads all values through this connection. A simplified overview of the motherboard architecture can be seen in Figure 3.3. This architecture makes this an integrated hardware component of the system. Since this controller only provides data though the interface IPMI or Redfish, we use the native data source and, therefore, satisfy **Req-2**. The chip comes preinstalled with the motherboard and only the tools need to be installed. This aligns with **Req-3**. Both interfaces can be accessed via the command line and satisfy **Req-4**.

Netio PowerPDU 4KS:

The last power monitoring tool we include is a physical PDU which measures the power draw between the PSU and the electrical socket, as can be seen in Figure 3.4. Therefore, this makes this power monitoring tool and external, hardware tool. It measures the whole system power draw. It provides the measured power draw via an Ethernet connection, which makes the data easily accessible through an endpoint. This satisfies **Req-2** and **Req-4**. The physical setup requires the power cables from the PSU to be plugged into the PDU and a physical network connection to the local area network must be established.

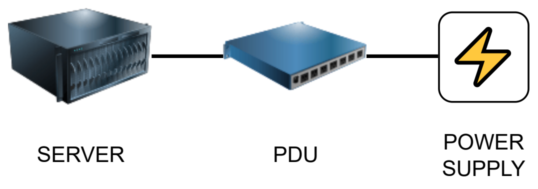


Figure 3.4: Power supply including the PowerPDU.

The PDU must be assigned to a static IP address. As this setup does not vary much with other external power monitoring tools, this aligns with **Req-3**.

Additional to the selection of tools, we further assessed different other power monitoring tools. An overview of all omitted internal tools, as well as the reason why we did not include them can be seen in Table 3.2. In this table, one can see that RAPL often provides the raw data which is further analyzed by other tools, which does not align with **Req-2** as we only consider tools which provide data directly without additional data processing. Furthermore, do other investigated tools like Advanced Configuration and Power Interface (ACPI) or PowerSensor3 require complex setup or specific requirements.

Name	**Req-2**	**Req-3**	**Req-4**
Scaphandre	Based on RAPL	✓	✓
PowerJoular	Based on RAPL	✓	✓
CPU Energy Meter	Based on RAPL	✓	✓
turbostat	Based on RAPL	✓	✓
powerstat	Based on RAPL	✓	✓
Perf	Based on RAPL	✓	✓
Intel® Performance Counter Monitor	Based on RAPL	✓	✓
ACPI	✓	Only possible with battery powered systems	✓
PowerSensor3	✓	Complex setup with hardware components	✓
Open Hardware Monitor	Based on RAPL	✓	×

Table 3.2: List of omitted power monitoring tools.

3.3 Benchmarking Tools

To analyze the power monitoring technologies under different conditions, we employ a set of benchmarking tools. First, we identify the system components that should be stressed during testing. Based on this, we define a set of requirements that suitable benchmarking tools must fulfill in order to be integrated into our framework and provide meaningful workloads. Finally, we present the selected tools that meet these criteria and are used throughout the evaluation.

3.3.1 System Components

We focus on four system components which are fundamental resource providers for a server. This list is derived from Ahmed et al. (12). In their publication the energy consumption is split up into five components, as presented in Formula 3.1.

$$P_{Server} = P_{base} + P_{CPU} + P_{disk} + P_{net} + P_{mem} \tag{3.1}$$

Where:

P_{Server}: is the total server power draw.

P_{base}: is the idle power draw.

P_{CPU}: is the CPU's power draw.

P_{disk}: is the storage power draw.

P_{net}: is the network interface power draw.

P_{mem}: is the memory power draw.

We relate our classification to this model and, therefore, split the benchmark tool selection into these four categories:

CPU – Memory – Network – Storage

3.3.2 Requirements

Given the large number of available benchmarking tools, a selection process is necessary. To guide this process, we define a set of requirements that candidate tools must fulfill. These requirements are derived from several factors: the architectural design of the benchmarking framework described in Section 3.1, the selected system components we want to execute benchmark experiments on, and the intended method of integration and interaction with the tools. The goal is to ensure compatibility, flexibility, and reproducibility within our framework. Based on these considerations, we define the following requirements:

Req-1: **Coverage**
The set of tools must be able to execute stress benchmarks on the different components established in Section 3.3.1.

Req-2: **Adjustability**
To execute different benchmark experiments on the same component, the tool should be adjustable in regards of workload and execution time.

Req-3: **Interface**
To be able to integrate the tool into the benchmarking framework, it must be accessible via the command line. Additionally, tools with graphical user interfaces (GUIs) are excluded, as they introduce unnecessary overhead and could interfere with the results.

Req-4: **Accessibility**
The tool should be openly accessible and free to use, to ensure that the benchmarking framework is easily reproducible in other research projects.

3.3.3 Tool Selection

We selected three benchmarking tools that collectively cover all system components outlined in Section 3.3.1, thereby fulfilling **Req-1**. The selected tools are listed in Table 3.3, along with the components they target. Additionally, the table includes excluded tools and the reasons for their omission. These selected tools form the basis for the implementation of the benchmarking experiment in our framework. As shown in the table, the tool selection addresses all relevant components targeted in our benchmark experiments. Moreover, all selected tools allow configurable benchmark execution, can be operated via the command line, and are freely available for use.

[1]https://github.com/ColinIanKing/stress-ng
[2]https://github.com/esnet/iperf
[3]https://github.com/axboe/fio
[4]https://www.spec.org/
[5]https://www.passmark.com/products/pt_linux/index.php
[6]https://github.com/akopytov/sysbench
[7]https://www.intel.com/content/www/us/en/developer/articles/tool/intelr-memory-latency-checker.html
[8]https://github.com/stressapptest/stressapptest
[9]https://www.userbenchmark.com/Software

Benchmarking Tool	**Component**	**Req-2**	**Req-3**	**Req-4**
stress-ng[1]	CPU, Memory	✓	✓	✓
iperf3[2]	Networking	✓	✓	✓
fio[3]	Storage	✓	✓	✓
SPEC[4]	CPU, Storage	Only fixed set	✓	Paid
PassMark Performance Test[5]	CPU, Memory	Not adjustable	✓	✓
sysbench[6]	CPU, Memory, Storage	Not adjustable	✓	✓
Intel® Memory Latency Checker[7]	Memory	Not adjustable	✓	✓
stressapptest[8]	Memory	Predefined	✓	✓
PC UserBenchmark[9]	CPU, Memory, Storage	Not adjustable	Not CLI	✓

Table 3.3: List of included benchmarking tools.

4

`BenchFrame` – Power-Benchmarking Framework

This section presents the implementation of `BenchFrame`[1], the power benchmarking framework described in Section 3. An overview of the framework's components is shown in Figure 4.1.

The goal of this section is to provide insight into the internal structure and technical decisions behind `BenchFrame`. Understanding the implementation is essential for assessing the reliability of the results, comparing the results to other research projects, and extending the framework for future use.

We begin by presenting the orchestration process, which manages the coordinated execution of the framework. Next, we introduce the monitoring process and highlight implementation details about the monitoring tools and the `/results` directory. Finally, we describe the implementation of the benchmarking experiments, covering both the execution logic and the individual scripts located in the `/benchmark` directory.

4.1 Orchestration Process

The orchestration process, realized as a python project, is divided into three steps. First, the monitoring process is started. In this step, all required resources are initialized, and the readings provided by the monitoring tools are collected and stored. Next, the benchmarking scripts can be executed. To ensure an undisturbed execution of the first two steps, both the monitoring and the script execution are run in subprocesses. After the script execution is completed, the monitoring process is terminated. At this point, all monitoring-specific

[1]The source code is available on GitHub: `https://github.com/THWU0412/BenchFrame`.

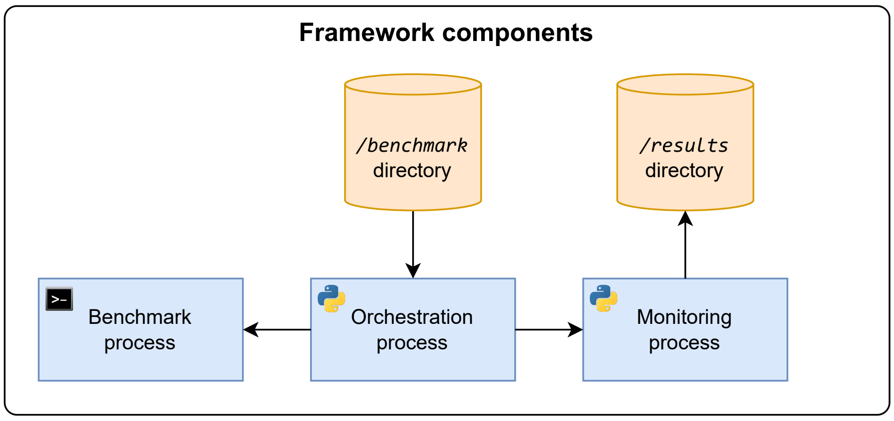


Figure 4.1: Overview of the components comprising the `BenchFrame` framework.

resources are cleaned up to ensure consistent conditions across runs. Finally, the monitoring data is processed. This step includes data preparation, diagram creation, and a statistical analysis.

4.2 Monitoring Process

The monitoring process, a python process triggered by the orchestration process, encompasses the creation of the necessary monitoring resources, and collecting and storing the data provided by the power monitoring tools.

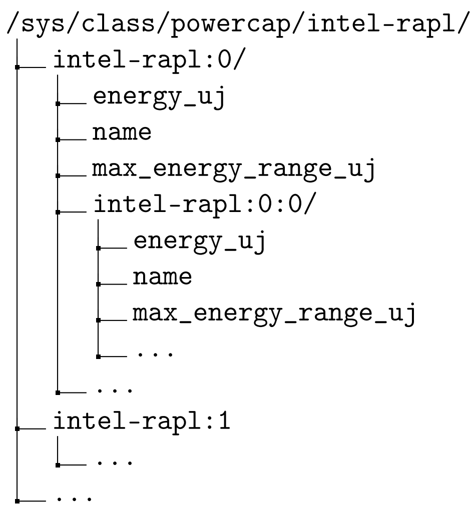
After creating the necessary monitoring resources, the process retrieves the power readings from the monitoring tools. It stores this data in a `.csv` file. For each new benchmark run, a new file is generated. Based on a preset granularity level, the amount of readings per second can be adjusted. The final collection of result files holds the raw data provided by the tools.

For each tools, the monitoring implementation varies. Therefore, we highlight the tools-specific implementation in `BenchFrame`.

RAPL

RAPL provides data through a power capping framework[1]. Access to this framework is provided through the powercap directory. An excerpt of the directory structure of powercap in Linux is presented in Figure 4.2.

[1]https://docs.kernel.org/power/powercap/powercap.html



```
/sys/class/powercap/intel-rapl/
    intel-rapl:0/
        energy_uj
        name
        max_energy_range_uj
        intel-rapl:0:0/
            energy_uj
            name
            max_energy_range_uj
            ...
        ...
    intel-rapl:1
        ...
    ...
```

Figure 4.2: Directory structure of the RAPL interface in the Linux `/sys` filesystem.

```python
rapl_path = '/sys/class/powercap'
rapl_sockets = sum(
    1 for name in os.listdir(rapl_path)
    if name.startswith('intel-rapl') and name.count(':') == 1
)
```

Listing 4.1: Analysis of the power capping framework.

The framework provides information per power zone, where each power zone represents one processor. In the presented file structure, the power zone is represented as `intel-rapl:0`. Various details can be accessed for each power zone. This directory includes, among other data, the power zone name, the energy counter `energy_uj` (in microjoules), and the maximum value `max_energy_range_uj` for this counter. Additionally, each power zone includes a subzone, such as `intel-rapl:0:0`, which represents the memory domain associated with the processor. This directory follows a similarly structure and also includes the name, the energy counter, the maximum counter value, and other related information.

The first step of the monitoring process is to setup all required resources. For RAPL, we first need to evaluated the structure of the power capping framework – this can be seen in Listing 4.1. For this purpose, we count the number of power zones in the `intel-rapl/` directory.

To retrieve the data from each power zone and the associated subzone, we first construct the path to each zone and then read the data. We iterate through each power zone and then

```python
def read_rapl(sockets):
    for socket in range(sockets):
        socket_paths = [f"/sys/class/powercap/intel-rapl{socket}/
            energy_uj",
                  f"/sys/class/powercap/intel-rapl:{socket}:0/
                    energy_uj"]
        with open(socket_paths[0], 'r') as file:
            data_rapl.append(int(file.read().strip()))
        with open(socket_paths[1], 'r') as file:
            data_rapl.append(int(file.read().strip()))
    return data_rapl
```

Listing 4.2: Retrieving RAPL readings per power zone.

return the collection of readings. This method returns for each processor and associated memory domain the consumed energy since the last reset.

This reading represent the energy (in microjoules). To convert this to power (in microwatts), we divide the difference between the last and the current reading through the time difference:

$$P = \frac{dE}{dT} = \frac{E_i - E_{i-1}}{T_i - T_{i-1}} \tag{4.1}$$

Where:

P: is the power draw.

E_i: is the current energy value.

P_{i-1}: is the previous energy value.

E_i: is the current timestamp.

E_{i-1}: is the previous timestamp.

IPMI/Redfish

For the implementation of `BenchFrame`, we utilize Redfish to retrieve power readings from the BMC, as it is a more modern and more secure implementation of IPMI.(13) Furthermore, it provides a python library[1] which we use to access power monitoring data from the RESTful API interface.

During the initialization phase of the monitoring step we need to create a Redfish object `REDFISH_OBJ` by calling the `redfish_client()` method and logging in. This happens in the method `setup_Redfish()` as presented in Listing 4.3.

[1]https://github.com/DMTF/python-redfish-library

```python
import redfish

def setup_Redfish():
    credentials = ("<<username>>", "<<password>>")
    url = "<<redfish_url>>"
    REDFISH_OBJ = redfish.redfish_client(base_url=url, username=
        credentials[0], password=credentials[1], default_prefix='/
        redfish/v1/')
    REDFISH_OBJ.login(auth="session")
    return REDFISH_OBJ
```

Listing 4.3: Redfish data retrieval

```python
import redfish

def read_Redfish(REDFISH_OBJ):
    response = REDFISH_OBJ.get("/redfish/v1/Chassis/1/Power")
    if response.status == 200:
        return response.dict['PowerControl'][0]['PowerConsumedWatts
            ']
```

Listing 4.4: Redfish data retrieval

Next, we can retrieve data through this object by calling the `get()` method with the corresponding url. To retrieve power information we can call the endpoint `/redfish/v1/Chassis/1/Power`. This gathers all data related to the power supply units and power consumption retrieved by the BMC. After we check whether the call was successful, we store the data. Hereby, we focus on the `PowerConsumedWatts` value, which includes the total power draw of the system.

After we finish the monitoring process, we need to clean up the resources by executing `REDFISH_OBJ.logout()`.

Netio PowerPDU 4KS

Netio PowerPDU provides data through several networking protocols like HTTP, MQTT, Modbus, and more. We utilize their python library `Netio`[1], which uses HTTP to access data from the PDUs.

Before accessing any power readings, we must initialize `Netio` object. To do so, we provide the IP address related to the PDU, as well as authentication credentials, which we

[1]https://github.com/netioproducts/PyNetio

```python
from Netio import Netio

def setup_PDU():
    PDU_L = Netio("<<NETIO_IP_L>>",
        auth_rw=("<<username", "<<password>>"))
    PDU_R = Netio("<<NETIO_IP_R>>",
        auth_rw=("<<username", "<<password>>"))
    return PDU_L, PDU_R
```

Listing 4.5: Initializing Netio objects.

```python
from Netio import Netio

def read_PDU(PDU_L, PDU_R):
    output_L = PDU_L.get_output('<<PDU_NODE_ID>>').Load
    output_R = PDU_R.get_output('<<PDU_NODE_ID>>').Load
    return {output_L, output_R}
```

Listing 4.6: Retrieving power data.

set during the installation of the PDUs. Since we work with a dual power supply setup, we issue two objects, as presented in Listing 4.5.

Afterwards, we can retrieve power readings by calling the `get_output()` method. Here we need to specify the ID related to the specific socket on the PDU. From the returned object we can read the `Load` attribute which gives us the current power draw in Watts. This step is presented in Listing 4.6

4.3 Benchmark Scripts

To execute the benchmark experiments, we utilize three benchmarking tools. These tools must be installed on the system and added to the path, so that we can include them in the benchmarking scripts. The scripts are `bash` scripts, which are executed by the orchestration process. Depending on the benchmark experiment, we modify the script settings to fit the purpose. In this section we give an introduction to each benchmarking tool and explain how we included it in the framework.

The benchmark scripts are collected in the benchmark directory. The orchestration process iterates over each script in this directory and, after starting the monitoring process, executes the script. This execution happens in a subprocess. The script execution

```python
from Netio import Netio

def run_script(run):
    script_path = f"/<<FULL_BENCHFRAME_PATH>>/{run[1]}"
    process = subprocess.Popen(['bash', script_path],
        stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
    process.wait()

for folder_item in os.listdir('scripts/'):
    run = (os.path.splitext(folder_item)[0], f"scripts/{folder_item
      }")
    run_script(run)
```

Listing 4.7: Execution of multiple benchmark scripts.

```bash
#!/bin/bash

stress-ng --cpu $(( $(nproc) / 2 )) --timeout 30s
```

Listing 4.8: Execution script for the CPU_HALF_100 benchmark.

orchestration can be seen in Listing 4.7.

stress-ng

We utilize the `stress-ng` benchmarking tool to execute stress tests on both the CPU and memory.

For CPU stress testing, we configure the number of CPUs cores designated to run the workload, as well as the load on each core. The benchmarking duration is specified using the `-timeout` flag. An example configuration for the `CPU_HALF_100` benchmark is shown in Listing 4.8.

In additional, we use `stress-ng` to execute memory benchmark test. For this purpose, we leverage the virtual memory feature, which allows us to define memory jobs, specify a job method, and specify the amount of memory to be used. An example configuration for the `MEM_READ` benchmark is provided in Listing 4.9.

iPerf3

For networking benchmarks, we utilize the `iPerf3` benchmarking tool. This tool can operate in two different modes: as a client or as a server. The system running the benchmarking

```bash
#!/bin/bash

stress-ng --vm 2 --vm-bytes 75% --vm-method read64 --timeout 30s
```

Listing 4.9: Execution script for the MEM_READ benchmark.

```bash
#!/bin/bash

ssh -i <<path to remote ssh key>> <<user>>@<<remote server>> "
    iperf3 -s -1" &

# Wait for the server to start
sleep 2

iperf3 -c <<remote ip address>> -t 30
```

Listing 4.10: Execution script for the NETWORK_SEND benchmark.

framework acts as the client, while a separate remote system functions as the server. The client is responsible for executing commands both locally and remotely. An example execution of the `NETWORK_SEND` benchmark is shown in Listing 4.10.

fio

The final benchmarking tool we utilize is `fio`, which enables stress testing of a system's storage. Since this tool operates at the file system level, it requires specifying test file path, the workload size, buffer size, operation type, and other parameters. An example configuration for the `STORAGE_WRITE` benchmark is shown in Listing 4.11

```
#!/bin/bash

fio --filename=tmp/testfile --size=2G --bs=4k --rw=write --direct=1
    --runtime=30s --time_based --ioengine=posixaio
```

Listing 4.11: Execution script for the STORAGE_WRITE benchmark.

5

Benchmark Design

We utilize `BenchFrame`, introduced in Section 4, to measure the energy consumption of the server under various workloads. By doing so, we can evaluate the power measurement tools, and answer **RQ-3** as presented in Section 1.2. To this end, we conduct 12 experiments on the system, which are classified into five different categories. The first four categories focus on the system component they address, while the fifth focuses solely on the performance and behavior of the power measurement tools. An additional idle measurement is used as a baseline for comparison. In the following sections we will first establish experiment objectives which we want to address with the experiments. Afterwards, we present an overview of the benchmark experiments we execute, as well as, the experiment setup. Then, we introduce the experiment methodology, and, finally, evaluate the results and present our key findings.

5.1 Evaluation Objectives

We classify the energy consumption benchmark experiments in five categories. Each category consists of benchmark test that are executed using the presented benchmark framework. To determine the most relevant server components, we follow an approach presented by Ahmed et al.(12) as presented in Section 3.3.1. Based on their model, we aligned the first four experiment categories with the relevant components. Therefore, we run benchmark experiments on the CPU, the RAM, the storage, and the Network Interface Controller (NIC). For each of these experiment categories we want to evaluate following experiment questions:

EQ-1: To what extent does the component influence the total energy consumption?

EQ-2: What are common workloads for the different components?

Category	Benchmark	Description
CPU	Static	Constant full CPU load across all cores
	Distributed	All CPUs running at 50% utilization vs. half running at 100% utilization
	Linear	Gradually increasing CPU usage
RAM	Static	Constant memory access, with read and write mixed
	Read vs. Write	One write job compared to one read job
NIC	Static	Constant data transmission to a remote server
	Send vs. Receive	One send job compared to one receive job
Storage	Static	Constant data writing to a file in storage
	Read vs. Write	Writing data to compared to reading data from a file
Tool	Latency	Tool behavior under rapidly changing loads
	Granularity	Tool reading in various granularity levels
	Stability	Error and spread over an extended monitoring period

Table 5.1: Overview of the benchmarking categories and their associated experiments.

EQ-3: How does the energy consumption vary under different workloads?

The fifth category investigates the behavior of the power monitoring tools under different settings. We determined the different experiments of this section by focusing on the important characteristics of the monitoring tools. Therefore, we decided on **latency**, **granularity**, and **stability**, as these answer the most important questions regarding the tool selection:

EQ-4 How accurate is the tool?

EQ-5 How fine can we measure with the tool?

EQ-6 How stable are the readings the tool provides?

5.2 Experiment Design

In this section we will present the experiments we conduct per category. For the first four categories, we compare a comprehensive workload benchmark run to the idle state of the system to address **EQ-1**. For each component we will then address **EQ-2** and establish common utilization patterns. These patterns are compared to answer **EQ-3**. For the last category – *Tool assessment* –, we execute one experiment per evaluation objective. An overview of the categories and the corresponding experiments is presented in Table 5.1.

5.2.1 CPU Benchmarks

To evaluate the influence of the CPU on the energy consumption of the whole system, we execute three CPU-intensive benchmark tests. The first experiment involves running a CPU-intensive workload on every core, allowing us to analyze how CPU workloads impact system-wide energy consumption and address **EQ-1**. To answer **EQ-2**, which focuses on workload patterns, we execute two additional experiments. These experiments examine how the distribution of workloads across CPU cores and linear growing CPU load affect the system's energy consumption. In total, the following three benchmark experiments are executed:

1. **Static**: This test aims to compare full CPU utilization with the idle state of the server. `BenchFrame` evaluates the influence of the CPU on the energy consumption with this experiment. The goal of this experiment is to evaluate the portion of energy that is consumed by the CPU intensive tasks.

2. **Distribution**: In this experiments we want to compare in which extend the distribution of workloads on the CPU cores influences the energy consumption. Therefore, `BenchFrame` includes these two benchmarks:
   - The first run stresses half of the CPU cores to full compute capacity. With this case we want to reproduce a fully saturated server.
   - The second test run stresses all CPU cores only to 50% of there total compute capacity.

   Based on these two test runs, we imitate similar workloads with two different distributions and analyze how the power consumption varies under these different conditions.

3. **Linear**: The last CPU-related experiments investigates the distribution of energy consumption under growing workload. For this purpose, `BenchFrame` linearly increments the workload put on all CPU cores.

5.2.2 Memory Benchmarks

By executing benchmark test on `BenchFrame` that execute memory stress tests, we investigate the influence on memory tasks on the total energy consumption of the system. To investigate the influence of memory workloads on the system, `BenchFrame` includes a static experiment. This experiments addresses **EQ-1** and analyses the influence of memory workloads on the system. As memory can be accessed either to read or to write data, `BenchFrame` includes a second experiment which compares these two functionalities. Therefore, we execute following memory benchmark experiment:

1. **Static**: `BenchFrame` compares a mixture of read and write operations to the idle state of the system. Hereby, we evaluate the influence of memory workloads on the system's energy consumption. We utilize 75% of the available memory capacity.
2. **Read vs. Write**: To examine the energy consumption behavior of memory access jobs, `BenchFrame` includes two different benchmark tests. The first analyzes write operations on the memory, while the second focuses on read operations. For both benchmark test, one job is issued per CPU core. With this experiment, we analyze the influence of memory access on the system's total energy consumption.

5.2.3 Network Benchmarks

To evaluate the influence of networking tasks through the utilization of the NIC, `BenchFrame` includes networking benchmarks. Hereby we evaluate in which degree networking jobs influence the energy consumption of the system. Therefore, `BenchFrame` executes following networking benchmark experiment:

1. **Static**: `BenchFrame` executes a send load on the NIC to evaluate the portion of energy the is necessary to execute networking tasks. For this purpose `BenchFrame` spawns a job which constantly sends data to a remote server.
2. **Send vs. Receive**: To compare the behavior of energy consumption between send and receive workloads, `BenchFrame` includes one experiment per workload type.

5.2.4 Storage Benchmarks

The last component-oriented category focuses on benchmark test related to storage. We evaluate the influence of storage tasks on the energy consumption of the system through these experiments. To do so, `BenchFrame` executes following benchmark experiments:

1. **Static**: To evaluate the influence of storage operation on the system's energy consumption, `BenchFrame` executes a mixed read and write experiment, where randomly data is either written or read from a file in storage.
2. **Read vs. Write**: To compare read and write operations, `BenchFrame` executes two experiment: First, `BenchFrame` writes a certain amount of data to a file in storage.

Afterwards, in a second benchmark test, `BenchFrame` reads this data again. With this experiment we compare the behavior of read and write jobs within the system.

5.2.5 Tool Benchmarks

The last category of benchmark experiments focuses on functionality and operation of the power monitoring tools. For this purpose, we execute experiments to evaluate how the tools functions under various settings and we investigate specific properties. The experiments we execute are:

1. **Latency**: To evaluate the response time of the monitoring tools, `BenchFrame` runs a rapidly changing CPU stress benchmark and we analyze how quickly the tools react to the changes in energy consumption. With this experiment we aim to highlight potential monitoring latencies of the tools.

2. **Granularity**: We compare different monitoring granularity settings to analyze how frequently data can be retrieved from the tools. For this purpose, we monitor the energy consumption with varying monitoring granularity and then analyze how often readings can be made under this setting.

3. **Stability**: To analyze the stability of the monitoring tools, we measure the energy consumption during an extended idle period and evaluate how much the readings fluctuate.

5.3 System Setup

An overview of the System Under Test (SUT) can be seen in Figure 5.1. ① **NODE** represents the main server which runs the benchmarking framework and executes the experiments. This server includes a 20-core Intel® Xeon® Silver 4416+ CPU, 8 × 32 GB DDR5 (SK Hynix HMCG88MEBRA107N) memory sticks which provide 256GB of RAM, a 1.92 TB Samsung PM893 SATA SSD, and 2 × Intel Ethernet Controller I210. These components are the ones addressed by the benchmarking experiments. Additionally, the server consists of a Supermicro X13SEW-F motherboard and runs on Ubuntu 22.04.4 LTS. The power supply for this server are two Supermicro PWS-861A-1R. Based on this dual power supply we include two ② **PDUs** in the system. Both are of the same type, namely *NETIO PowerPDU 4KS*. The server's power supply units are connected to the PDUs via ③ industry standard power cables. And the measurement from the PDUs can be retrieved

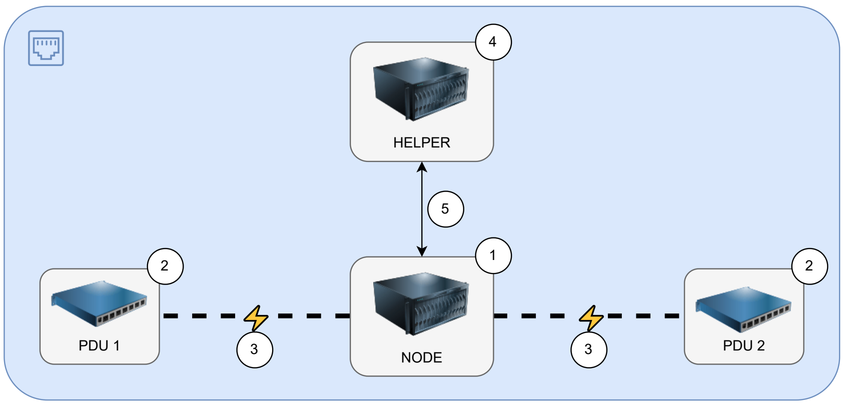


Figure 5.1: System architecture used for the benchmarking experiments.

via the local Ethernet network. Additionally, a ④ **HELPER** server is installed to function as a communicator for networking experiments. This second server includes 2 × Intel® Xeon® Silver 4210R, 2 × Intel Ethernet Controller I210, and runs on Ubuntu 22.04.5 LTS. The two servers are connected via a ⑤ private local Ethernet network.

5.4 Benchmark Methodology

The benchmark experiments were executed on the hardware described in Section 5.3, using the framework presented in Section 4. Each experiment consist of at least one benchmark run. The benchmark run execution always follows the same sequence:

1. The experiment is executed five times and the results are monitoring and stored.
2. After the experiment finishes, the results are loaded, and the average per second is calculated. This results in a dataset where either zero or one value is present per second.
3. Missing values are estimated using *linear interpolation*.
4. The resulting data, consisting of one monitoring result per second,[1] is stored. Each result set therefore contains five equally long data collections, each uniformly structured with one reading per second.

[1]This value may vary depending on the monitoring granularity.

Through this process, we eliminate distortions caused by faulty measurements or other processes and can generate a clean dataset. To address the experiments presented in Section 5.1, we execute various benchmark runs. The complete benchmark suit is presented in Appendix B.

6

Evaluation

After executing the experiments and collecting the results we evaluate the measurements. Hereby we start with a comparison of the static benchmark experiments. For each workload type, one static benchmark experiments was executed. After this we present the results for the component specific experiments. At last, we highlight our results from the tool behavioral experiments.

6.1 Static Evaluation

To analyze the influence of different workloads on the energy consumption of the system, we compare the total energy consumed during the experiment run. Therefore, we compare the static experiment, where a constant component-specific load is put on the system. We analyze the total energy consumption recorded during these runs to assess the component-specific impact on overall power usage. To make these comparable, we designed the experiments to run for the same duration and put similar workloads on the system. The total energy consumption of the static workload experiments are presented in Figure 6.1.

Through these experiments we were able to show that CPU workloads have the most influence on the energy consumption of the system. In our experiments the energy consumption of the CPU benchmark compared to the idle state increased by a factor of 2.51 for RAPL, 1.66 for Redfish, and 1.28 for the Netio PowerPDU.

Observation 1

CPU workloads have the greatest impact on the system's energy consumption, followed by memory workloads. In contrast, storage and networking workloads exhibit only a minor influence.

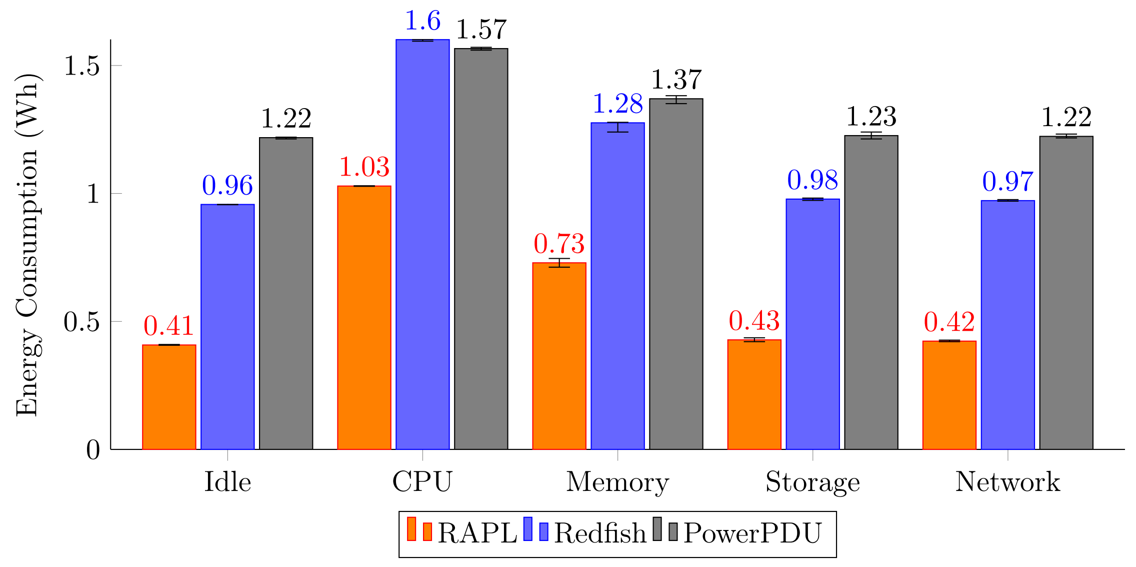


Figure 6.1: Average total energy consumption per static experiment with error bars (30-second active benchmark duration, excluding setup and processing time).

6.2 Component Workload Evaluation

Additional to the static experiment evaluation, we executed component based experiments, specific to the component's common use cases. In this section we present the results for this analysis and highlight behavioral characteristic in the energy consumption and how they were detected by the different monitoring tools.

CPU Benchmarks

In the CPU benchmark category, as presented in Section 5.2.1, we execute two additional use-case-based experiments, namely **Distribution** and **Linear**.

<u>Distribution</u>
In this experiment, we investigate the relationship between energy consumption and different workload distributions across CPU cores. The total energy consumption for both experiments is shown in Figure 6.2. Across all three measurement sources, the results suggest that distributing a workload across all CPU cores is more energy-efficient than running a subset of cores at full utilization.

However, the CPU benchmarking tool may introduce inaccuracies, as it does not guarantee a fixed level of CPU utilization. To account for this, we analyze the efficiency of the two benchmark runs by calculating the average energy consumption per 1% of CPU utilization. This allows us to evaluate whether distributing workloads across multiple cores is more

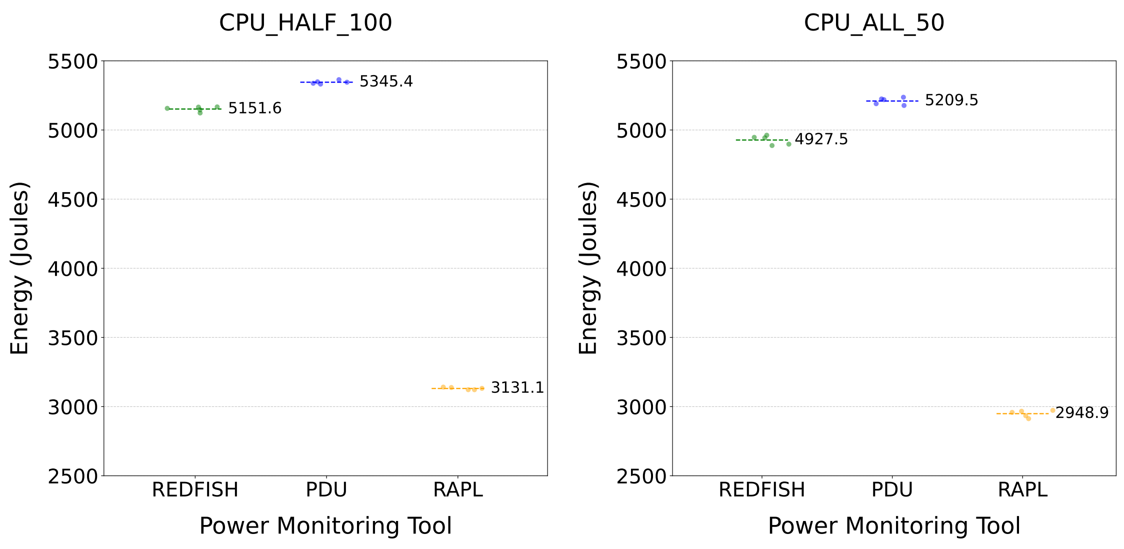


Figure 6.2: Comparison of full utilization on half of the CPU cores and 50% utilization on all CPU cores, showing individual runs and the overall mean.

energy efficient than fully utilizing a subset of cores before allocating load to additional ones.

Efficiency is calculated by dividing the total energy consumption by the average CPU utilization, as shown in Equation 6.1:

$$\mathit{Efficiency}_b = \frac{E_b}{U_b^{\text{CPU}}} \tag{6.1}$$

Where:

$\mathit{Efficiency}_b$: is the efficiency of benchmark b (in $J/\%$).

E_b: is the energy consumed during benchmark b (in J).

U_b^{CPU}: is the average CPU utilization during benchmark b (in %).

To calculate efficiency, we first retrieve the average CPU utilization from the monitoring data: for the `CPU_HALF_100` benchmark, we observe **44.30%**, and for the `CPU_ALL_50` benchmark, **43.35%**. These values represent the average CPU utilization recorded over the entire duration of the benchmark run.

Additional, we retrieve the total energy consumption over the benchmark run. This metric as well as the resulting efficiency are presented in Table 6.1.

These results support our hypothesis that distributing workloads across multiple cores leads to higher energy efficiency compared to fully utilizing fewer cores. Furthermore, this

Tool	CPU_HALF_100		CPU_ALL_50	
	E_b	Efficiency (J/%)	Mean energy (J)	Efficiency (J/%)
Redfish	5151.6	116.29	4917.5	113.44
PDU	5345.4	120.66	5209.5	119.9
RAPL	3131.1	70.68	2948.9	68.02

Table 6.1: Efficiency of each distribution modes.

behavior is consistently observed across all power monitoring tools, indicating that each tool reliably detects this pattern.

Observation 2

Distributing workloads across multiple CPU core leads to a more energy efficient execution compared to utilizing single cores to full extend before including additional cores. All three monitoring tools are able to observe this pattern. This shows that the energy consumption of CPU cores is not linearly related to the CPU utilization.

Linear

To further investigate the energy consumption of CPU cores under varying utilization levels, we analyze the results from the `CPU_LINEAR` benchmark, where the CPU utilization across all cores is consistently increased by 10% till full utilization. The energy readings from all three power monitoring tools under linearly increasing CPU utilization are shown in Figure 6.3.

Two notable patterns emerge from the results. First, energy consumption increases most rapidly between approximately 20% and 50% CPU utilization. Second, the power readings from Redfish and RAPL increase more steeply than those from the PowerPDU – so much so that, at around 80% CPU utilization, the Redfish readings exceed those of the PowerPDU.

Observation 3

The power consumption does not increase linearly with CPU utilization. Furthermore, the readings from Redfish and RAPL rise more quickly than those from the PowerPDU.

Memory Benchmarks

In addition to the static memory benchmark, we execute one more experiment where we compare the influence of read and write operations on the system's energy consumption.

Read vs. Write

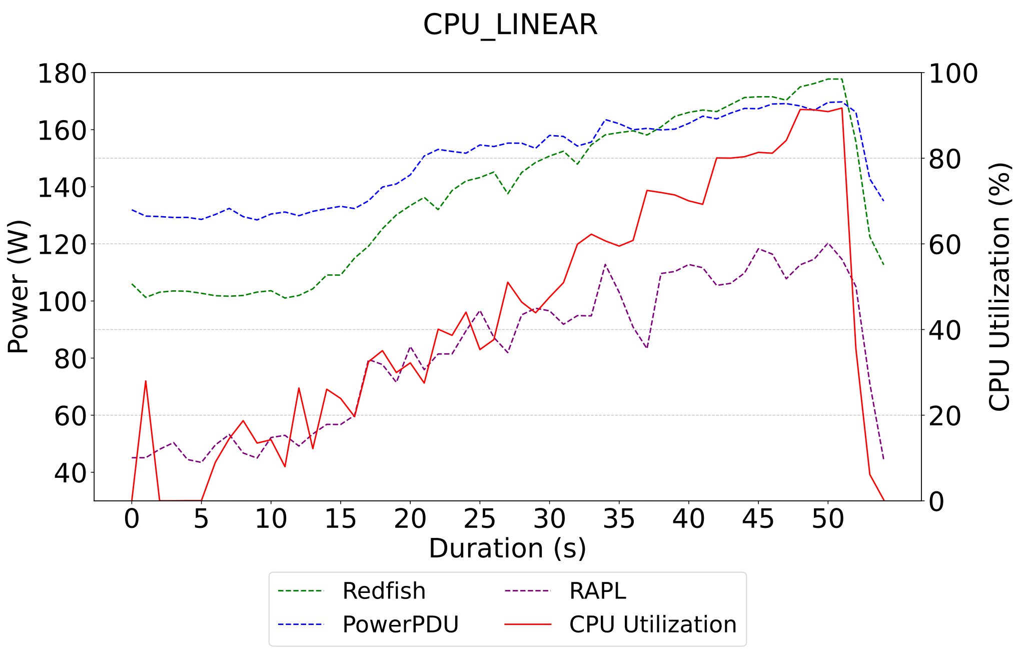


Figure 6.3: Average power draw under linear increasing CPU utilization.

In this experiment, we investigate the energy consumption of memory read operations compared to write operations. The results for both benchmarks are presented in Figure 6.4.

In both workload modes, the power readings are similarly distributed, suggesting comparable energy consumption patterns. However, when analyzing the total energy consumption, we observe that read operations consume slightly more energy than write operations. The average total energy consumption for read and write workloads is 1.41kWh and 1.36kWh for Redfish, 1.53kWh and 1.49kWh for the PowerPDU, and 0.79kWh and 0.76kWh for RAPL, respectively.

Furthermore, we find that overall memory utilization does not significantly influence the system's energy consumption. This is evident in the diagrams, where an increase in memory usage does not correspond to a noticeable change in power draw. The frequency of memory access has a greater impact on energy consumption.

Observation 4

Read operations tend to have a bigger influence on the system's energy consumption compared to write operations. Additionally, the memory access rate influences the energy consumption more than the utilization of the total memory.

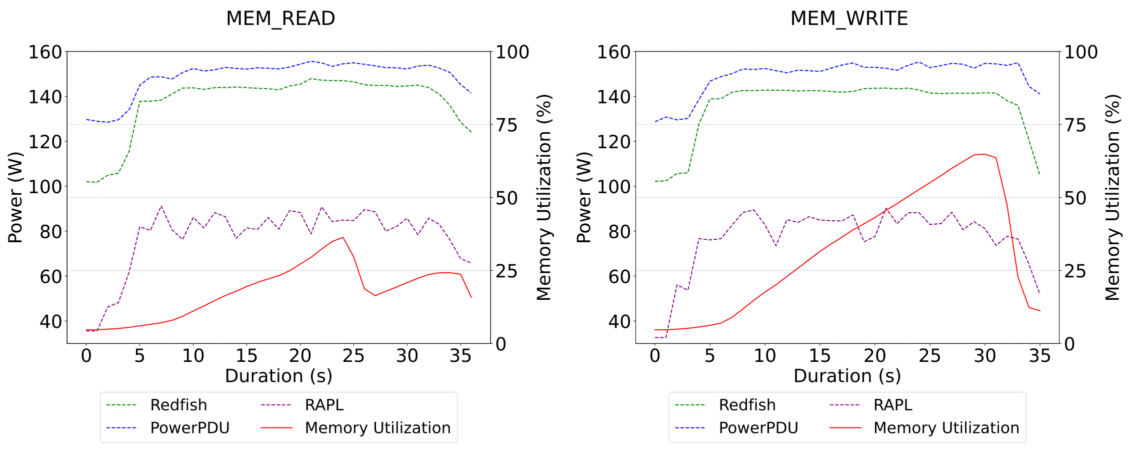


Figure 6.4: Power readings from read and write workloads on the memory.

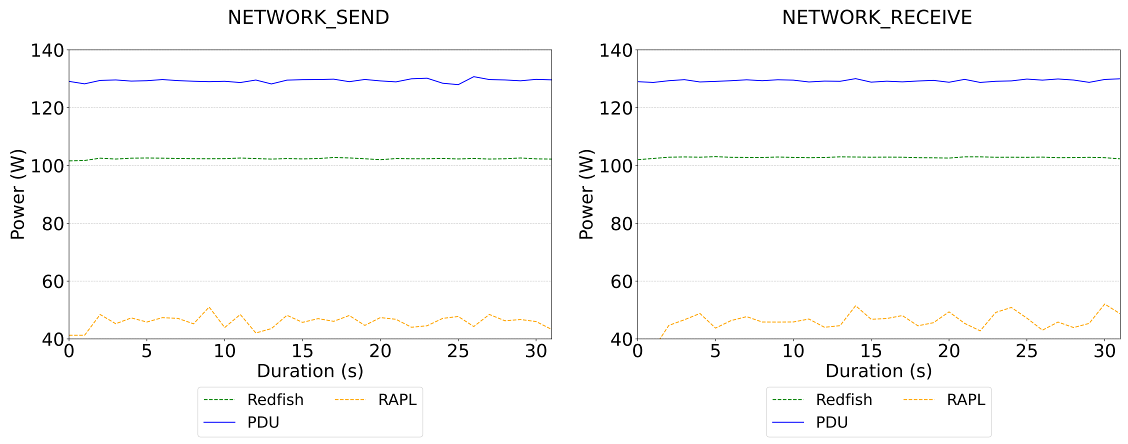


Figure 6.5: Power draw during send and receive benchmarks.

Network Benchmarks

In networking sending or receiving data are the only two different modes. Therefore, we set out to investigate differences in these workload types. But, as was already witnessed in the static analysis in Section 6.1, networking has only little influence on the system's energy consumption. Therefore, we do not expect to see huge differences in those two benchmark experiments. The result, presented in Figure 6.5, confirm this assumption and show that neither sending or receiving data has a huge influence on the system's energy consumption.

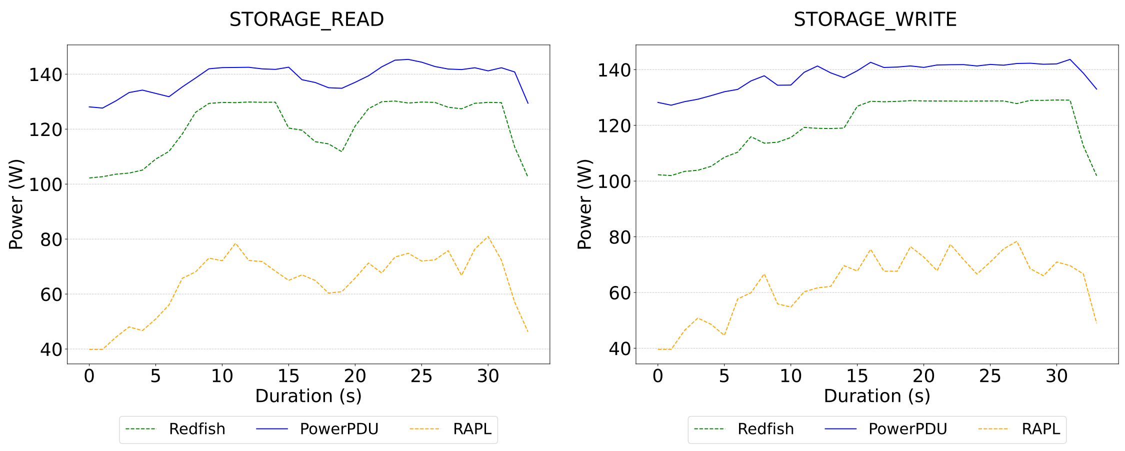


Figure 6.6: Power readings from read and write workloads on the storage.

Storage Benchmarks

The last component based benchmark experiment is related to read and write operations to the storage.

Read vs. Write

The power readings of read and write workloads for all three tools is shown in Figure 6.6.

As the diagrams show, the power readings are not constant for either benchmark. However, when analyzing the total energy consumption, we observe that all tools report similar values for both read and write operations. The total energy consumption for read and write benchmarks is 1.14kWh and 1.13kWh for Redfish, 1.31kWh and 1.30kWh for the PowerPDU, and 0.60kWh and 0.58kWh for RAPL, respectively.

An additional important factor in storage operations is the bandwidth at which data is transferred. For write operations, the average bandwidth is 78.2MiB/s, while for read operations it reaches 126MiB/s. Consequently, despite comparable power consumption, read operations are able to process approximately 61% more data, making them more efficient in terms of data throughput.

Observation 5

Read and write operations exhibit similar energy consumption. However, read operations achieve higher bandwidth, enabling more efficient and faster data processing.

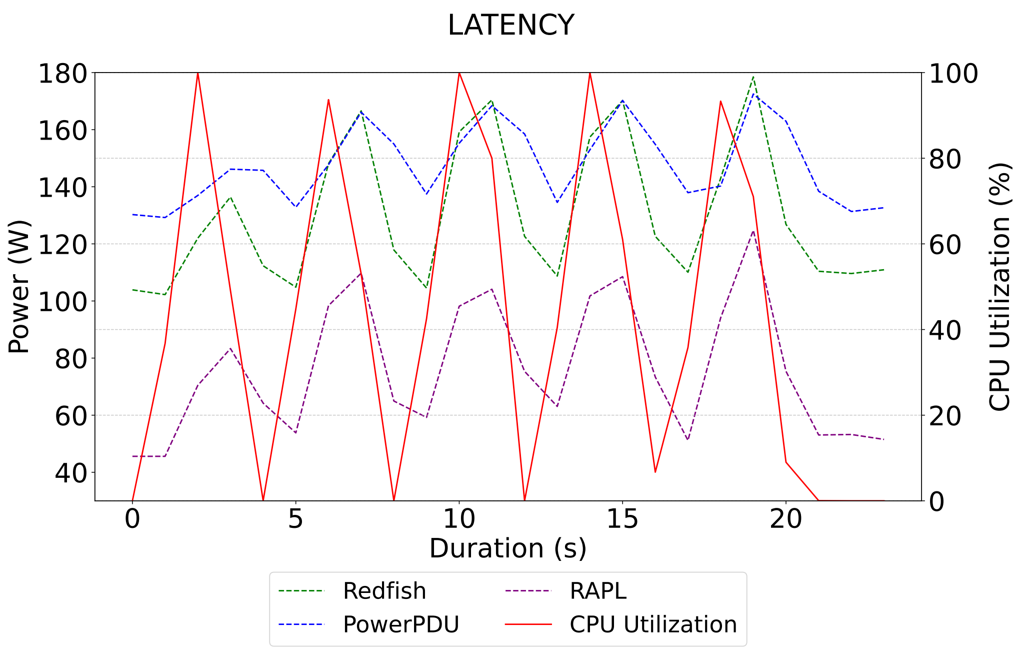


Figure 6.7: Latency benchmark plot including power readings and CPU utilization.

6.3 Usability Evaluation

Latency

To analyze the latency of the tools, we compare how quickly the readings adjust to changes in the workload. For this purpose we rapidly change the CPU utilization and compare the power readings. The results of this benchmark can be seen in Figure 6.7. Hereby, all power monitoring tools detect changes at least with a delay of a second. But especially the PowerPDU react on changes in the CPU utilization slower. This can be seen for example at timestamp 4, where Redfish and RAPL readings are already decreasing, while the readings from the PowerPDU do not change. This behavior can be identified throughout the whole benchmark experiment.

Observation 6

All investigated tools provide readings with at least one second delay. But compared to the other monitoring tools, the PowerPDU responds more slowly to changes in system utilization.

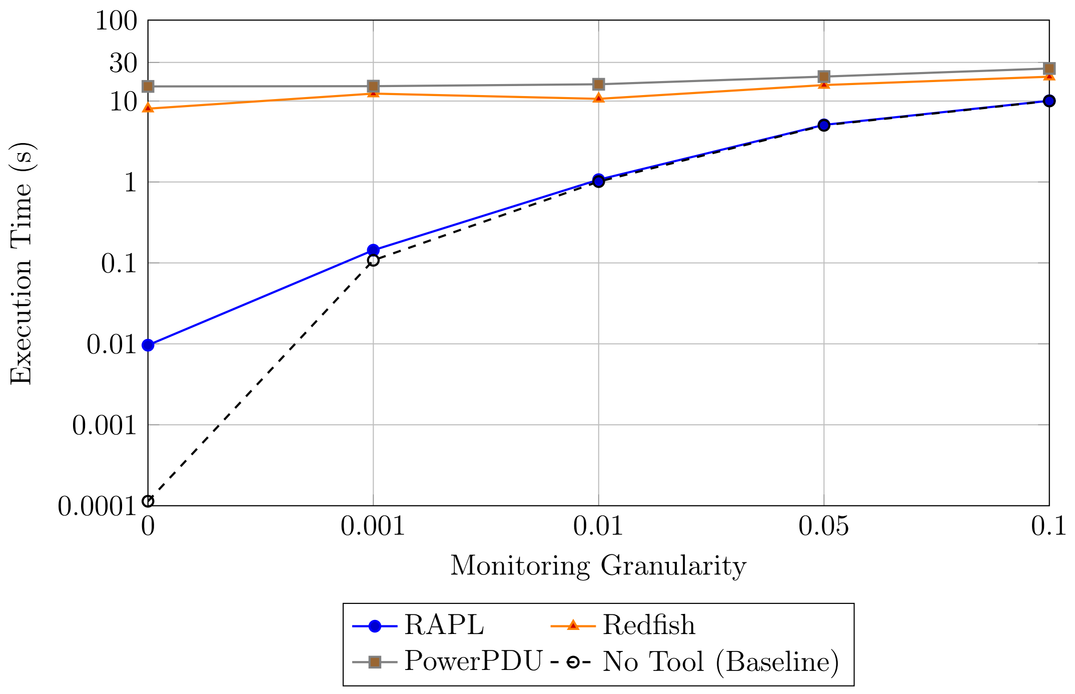


Figure 6.8: Log-scale comparison of execution times across different monitoring granularities.

Granularity

To investigate the granularity, we execute an additional benchmark test which is not part of `BenchFrame`. For this purpose, we utilize the additional code as described in Appendix C. Hereby we want to investigate how fast we can retrieve power readings from the tools. Our results are presented in Figure 6.8 and the data readings can be found in Appendix D. Between each new reading, we wait for a certain amount of time. This value is depicted as *Monitoring Granularity*. To calculate the execution time, we monitor the time before and after each run and calculate the difference. The run itself consists of 100 consecutive readings. The results presented are averaged over five runs. To have a comparison baseline, we also executed the script without retrieving any tool readings.

Observation 7

RAPL offers the most granular measurement capability. Since it is an internal software tool that reads directly from registers, no additional overhead from APIs or external communication is introduced. In contrast, the PowerPDU performs worst in the granularity experiment, likely due to networking overhead, as its data must be transmitted over the local area network.

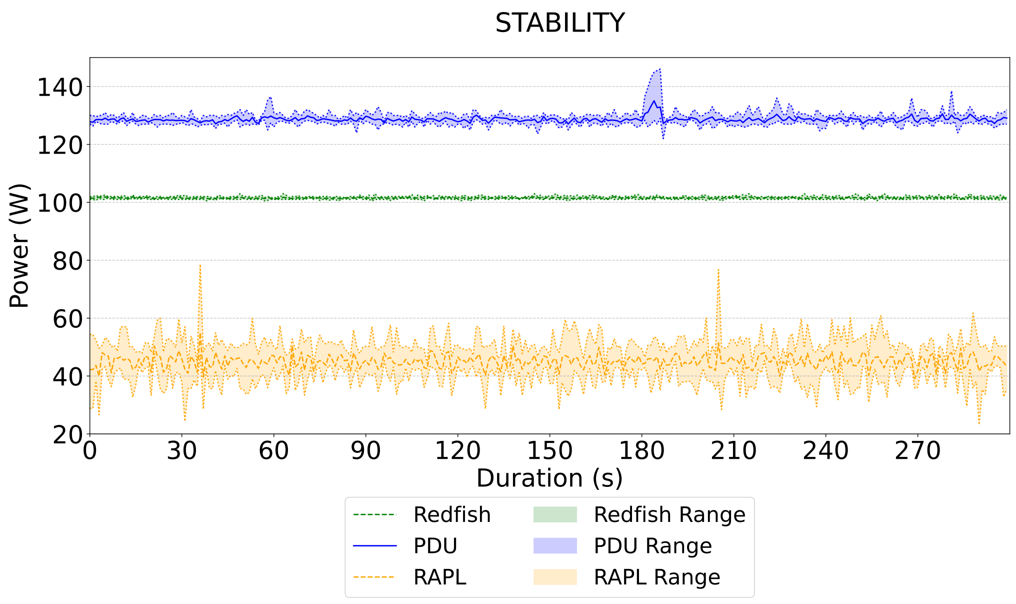


Figure 6.9: Stability benchmark which includes measurements over 300s without any workload on the system.

Stability

To analyze the stability of the readings provided by the different power monitoring tools, we examine the distribution of measurements during an idle period in which no workload is applied to the system. For this purpose, we calculate the mean power draw (μ) and the standard deviation (σ). The results are presented in Table 6.2.

The average power draw, measured in watts, is highest for the PowerPDU and lowest for RAPL, which aligns with the findings reported in Observation 1. However, the most stable readings are provided by Redfish, which exhibits a standard deviation of only 0.18. In contrast, the highest variability is observed in the RAPL readings, with a standard deviation of 2.24.

	RAPL	Redfish	Netio PowerPDU 4KS
μ (kW)	45.03	101.52	128.62
σ	2.24	0.18	0.85

Table 6.2: Mean and standard deviation of the stability experiment.

Observation 8

Redfish provides the most stable power readings, while RAPL exhibits the highest fluctuations.

6.4 Conclusion

The benchmark experiments focused on two main categories: component-based analysis and the operability of the monitoring tools. By formulating a set of experimental questions, we were able to, first, compare different system workloads. Second, we examined various workload patterns across different components and analyzed their impact on overall energy consumption. Our findings indicate that CPU workloads have the most significant influence on energy consumption, followed by memory and storage operations.

In addition, we evaluated the operability of the monitoring tools. We found that the PowerPDU exhibits the highest delay in response, while Redfish and RAPL provide more time-accurate readings. Furthermore, the PowerPDU offers the lowest measurement granularity, whereas RAPL supports the highest. Finally, in terms of stability, Redfish delivered the most consistent readings, while RAPL showed the greatest variability.

7

Related Work

Analyzing the energy consumption of servers is a well-established research field.[1] Therefore, we highlight recent advancements in this area and examine the current state of research. To this end, we present the most relevant research projects and compare them to our work. We analyze the selected projects with respect to the power monitoring tools employed and the benchmarking suit used. A comprehensive list of related research projects is provided in Table 7.1.

Reference	Year	Hardware and software tools	Internal and external tools	Component based benchmarking	Operability benchmarking
Hackenberg et al.(14)	2013	✓	✓	–	–
Khan et al.(15)	2018	–	–	–	–
Kavanagh et al.(16)	2019	–	✓	–	–
van Kemenade(17)	2024	✓	✓	–	✓
Freina(18)	2024	–	–	–	–
Andringa et al.(19)	2025	✓	✓	–	–
This thesis	2025	✓	✓	✓	✓

Table 7.1: Related research projects and their focus.

7.1 Power Monitoring Tools

The research projects by Khan et al.(15) and van Kemenade(17) focus exclusively on RAPL or RAPL-based tools. Both projects selected this tool due to it's popularity and ease of

[1]As of July 16th, 2025, *IEEE Xplore* listed 8,566 publications related to the query *"server energy consumption"* since 2020.

use, as it requires neither hardware installation or complex setup. However, limiting the analysis on a single tool prevents a comprehensive comparison and restricts all observations made during the project to RAPL alone.

On the other hand, focusing on a fine-grained, purely hardware-based evaluation – such as the approach presented by Hackenberg et al.(14) – requires additional installation and setup. While this method is feasible under research conditions, it lacks applicability in typical data center environments.

Our project includes both internal and external tools. This ensures comparability and allows us to identify similarities in the results. Furthermore, it enables us to draw conclusions across multiple tools and directly compare their behavior.

7.2 Benchmarking

The focus and scope of benchmarking in energy-related research can vary widely. While our approach targets system components and the operability of monitoring tools, other research projects conduct their evaluations using standard benchmark suites or specific use cases.

Freina(18) focuses his evaluation on estimating power consumption across the compute continuum – comprising thing, edge, and cloud resources. Hackenberg et al.(14) apply a variety of common benchmark tests in their evaluation. This makes their results more comparable to other studies using similar tests. However, this approach limits the potential for fine-grained analysis of the relationship between specific benchmark tasks and energy consumption.

By including both component-based and operability-focused benchmarks in our suite, we can make informed statements about the energy consumption of specific components under common workloads. Additionally, we are able to assess the functionality of monitoring tools and identify limitations inherent to particular tools.

8 Conclusion

As part of this research project, we investigated different power monitoring techniques. For this purpose, we designed and implemented a benchmark framework. This framework allows us to execute and analyze benchmark experiments to assess different power monitoring tools. We established a comprehensive benchmark suite which focuses on, first, common workload types and, second, on the operability of the tools.

8.1 Answering Research Questions

RQ-1: *Which techniques and tools are available to monitor server energy consumption and how do they differ?*
To answer the first research question, we investigated different tools, which are currently provided by either the system or external vendors. We established a classification which differentiates between the implementation method, the measurement scope, and the integration of different tools. Afterwards, we set up requirements for a comprehensive tool selection and decided on a set of tool which covers these requirements.
RQ-1: *How can power monitoring tools be evaluated?*
Research question two focuses on the assessment of the tools. For this purpose, we investigated common workload types for servers related to its components. We defined four component based benchmark categories, namely CPU, memory, networking, and storage benchmarks. Additional, we added operability benchmarks to investigate the functionality of the tool under different settings.
RQ-1: *How do different workloads influence the overall system's energy consumption, and to what extent can power monitoring tools capture this behavior?*
The final research question relates to the execution of the benchmarking experiments.

First, we designed a benchmarking framework which allows us to execute benchmark experiments. After successfully implementing the framework and executing it, we were able to assess the different power monitoring tools, based on the benchmark experiments. Our observations lead to a deeper understanding of the tools and their operability. Through various observations, we discovered that:

1. CPU workloads influence the energy consumption the most, followed by memory workloads.
2. All investigated power monitoring tools are able to document power patterns.
3. Software-based tools allow a more fine-grained monitoring.
4. Internal tools may only monitor a subset of components, while external tools take the total energy consumption into account.

8.2 Threats to Validity

Even though we implemented a comprehensive benchmarking framework, some parts of the implementation and experiment execution are subject to question. We identified following threats to validity:

1. Experiment isolation

   Isolating workloads without any interference is almost impossible on servers. Every server runs background jobs, which means that apart from the benchmark experiments other processes may stress the system. This can interfere with the power readings.

2. Workload comparability

   To analyze the influence a certain workload has on the system's energy consumption, we compared various workloads. The benchmark experiments were established using common attributes, recommended by the documentation of each tool. But to make the results fully comparable, first, the experiments itself should be compared. Hereby, the questions arises how different workloads can be compared, since all of them address different operation types.

8.3 Future Work

Based on our research advances, future projects should further investigate the tools. Therefore, we established a list of research directions:

1. One important aspect is the overhead of the different tools. This metric is not only important to better understand the power readings provided by the tools, but also influences the energy consumption of the system. Therefore, it influences research advances but also the application of tools in established data centers.

2. To make the results of this framework more easily comparable, additionally to the implemented benchmarks, already established benchmark suits should be tested. This allows us to compare not only the tools but also assess the framework for correctness, execution overhead, and other metrics

3. Additional to the component based analysis, the benchmark experiments should be expanded with use-case experiments. Our approach allowed us to investigate the influence on component based workloads, but often servers need to handle various jobs simultaneously.

4. Apart from the three monitoring tools are other methods to evaluate the energy consumptions of servers, which are simulations. This benchmark framework could be replicated in a simulation to estimated the energy consumption of systems without the need to establish a complete experiment setup.

References

[1] Simon Kemp. **Digital 2025: Global Overview Report**. Technical report, We Are Social, Meltwater, April 2025. 1

[2] Bernardo Betley, Hana Dib, Bjørnar Jensen, and Bernhard Mühlreiter. **The state of cloud computing in Europe: Increasing adoption, low returns, huge potential**. *McKinsey Digital*, April 2024. 1

[3] Steven Gonzalez Monserrate. **The Cloud Is Material: On the Environmental Impacts of Computation and Data Storage**. *MIT Case Studies in Social and Ethical Responsibilities of Computing*, **2022**(Winter), January 2022. 1

[4] Council of the European Union European Parliament. **Directive (EU) 2023/1791 of the European Parliament and of the Council of 13 September 2023 on energy efficiency and amending Regulation (EU) 2023/955 (recast) (Text with EEA relevance)**, September 2023. 1

[5] Klaus Pohl and Chris Rupp. *Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam, Foundation Level, IREB compliant.* Rocky Nook, Santa Barbara, CA, second edition edition, 2015. 6, 12

[6] Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan. *The Datacenter as a Computer: Designing Warehouse-Scale Machines.* Synthesis Lectures on Computer Architecture. Springer International Publishing, Cham, 2019. 9

[7] Corey Gough, Ian Steiner, and Winston Saunders. *Energy Efficient Servers: Blueprints for Data Center Optimization.* Apress, Berkeley, CA, 2015. 10

[8] Weiwei Lin, Fang Shi, Wentai Wu, Keqin Li, Guangxin Wu, and Al-Alas Mohammed. **A Taxonomy and Survey of Power Models and Power Modeling for Cloud Servers**. *ACM Computing Surveys*, **53**(5):1–41, September 2021. 14

[9] The kernel development community. **The Linux Kernel documentation**, 2025. 16

[10] Intel Corporation. **Intel® 64 and IA-32 Architectures Software Developers Manual**, 2025. 16

[11] Super Micro Computer, Inc. **IPMI User's Guide**, 2020. 17

[12] Kazi Main Uddin Ahmed, Math H. J. Bollen, and Manuel Alvarez. **A Review of Data Centers Energy Consumption and Reliability Modeling**. *IEEE Access*, **9**:152536–152563, 2021. Publisher: Institute of Electrical and Electronics Engineers (IEEE). 19, 33

[13] Distributed Management Task Force. **Redfish White Paper**, 2018. 26

[14] Daniel Hackenberg, Thomas Ilsche, Robert Schöne, Daniel Molka, Maik Schmidt, and Wolfgang E. Nagel. **Power measurement techniques on standard compute nodes: A quantitative comparison**. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 194–204, April 2013. 53, 54

[15] 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*, **3**(2):1–26, June 2018. 53

[16] Richard Kavanagh and Karim Djemame. **Rapid and accurate energy models through calibration with IPMI and RAPL**. *Concurrency and Computation: Practice and Experience*, **31**(13):e5124, 2019. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5124. 53

[17] Tim van Kemenade. *Real-time Scaphandre Energy Metrics Pipeline Integrated with Escheduler*. Master's thesis, Vrije Universiteit Amsterdam, Amsterdam, 2024. 53

[18] David Freina. *End-to-End Power Model for the Compute Continuum*. Master's thesis, Vrije Universiteit Amsterdam, Amsterdam, 2024. 53, 54

[19] Lars Andringa, Brian Setz, and Vasilios Andrikopoulos. **Understanding the Energy Consumption of Cloud-native Software Systems**. In *Proceedings of the 16th ACM/SPEC International Conference on Performance Engineering*, ICPE

'25, pages 309–319, New York, NY, USA, 2025. Association for Computing Machinery. 53

[20] NETIO Products a.s. **PowerPDU Manual**, 2024. 64

Appendix A

Reproducibility

In this section we present how `BenchFrame` can be downloaded and executed. Through this explanation, other researchers are able to reproduce our results. This section includes the setup of the framework, as well as the requirements of the benchmark experiments.

A.1 Artifact check-list (meta-information)

- **Program:** `BenchFrame`
- **Compilation:** Python3
- **Run-time environment:** Ubuntu 22.04.4 LTS, Python 3.10.12, root rights required
- **Hardware:** System with RAPL and Redfish access, Netio PowerPDU 4KS
- **Execution:** For execution, the required software and hardware must be installed. Next, the `template.conf` file must be copied, renamed to `host.conf` and filled out with all necessary information. Afterwards, the framework can be executed.
- **Metrics:** Energy consumption and power over time (RAPL, Redfish, PowerPDU), CPU utilization, memory utilization
- **Output:** One .csv file and two diagrams showing the power over time and the total energy consumed per experiment
- **Experiments:** See Appendix B
- **How much time is needed to complete experiments?:** 1.5h
- **Publicly available?:** Yes, under `https://github.com/THWU0412/BenchFrame`
- **Code licenses:** MIT license

A.2 Description

A.2.1 How to access

The source code of `BenchFrame` is shared via GitHub. Users can simply clone the repository and execute the framework directly, no further compilation or builds are needed.

A.2.2 Hardware dependencies

`BenchFrame` was built with a specific set of power monitoring tools. Therefore, the system must meet these requirements.
The tools are:

1. RAPL
2. Redfish
3. Netio PowerPDU 4KS

The user must ensure that these monitoring tools are installed and can be accessed.

A.2.3 Software dependencies

The power monitoring tools need to be installed as described in their manuals.
This includes:

1. RAPL: No additional software needed
2. Redfish: Enable IPMI as described here: `https://www.thomas-krenn.com/en/wiki/Configuring_IPMI_under_Linux_using_ipmitool`. This will also enable Redfish
3. PowerPDU: Installation as described in the manual(20)

Additional, several tools need to be installed to run the benchmarking experiments.
These are:

1. stress-ng (`https://github.com/ColinIanKing/stress-ng`)
2. iperf3 (`https://github.com/esnet/iperf`)
3. fio (`https://github.com/axboe/fio`)

A.3 Installation

To execute the framework, all software and hardware requirements must be met. Once the required software is installed, the framework can be executed. The framework must be executed with root access. Therefore, the power monitoring software and the benchmarking tools should be installed globally.

A.4 Experiment workflow

The experiments are executed successively. Before the experiment is executed, all necessary resources are created. During each experiment run various metrics are collected and stored. After the experiment completed, the resources are released again. Therefore, each experiment run is executed under the same conditions.

A.5 Evaluation and expected results

After the benchmark was executed and the framework terminates, a new folder including the results should have been added in the /results directory. This folder contains one .csv file per experiment. This .csv file includes the raw data collected during the experiment run. Additional, a folder called /cleaned should contain the cleaned and processed data and the plots of the experiment.

A.6 Experiment customization

The experiment suit can easily be extended. To add experiments to the suit, one simply needs to add a shell script in the /results folder of the project. The name of the file is utilized as an identifier and should be unique. The script should run on it's own and it should be executable by the root user.

Appendix B

Benchmark Suite

Name	Description	Related category
Idle	No workload on the system	All
CPU_STATIC	Stress all CPUs 100 % for 30s	CPU
CPU_ALL_50	Stress half of all CPUs 100 % for 30s	CPU
CPU_HALF_100	Stress all CPUs 50 % for 30s	CPU
CPU_LINEAR	Linearly increase the CPU load	CPU
MEM_STATIC	Random read and write operations on up to 75% of the memory for 30s	Memory
MEM_READ	Read operation on up to 75% of the available memory for 30s	Memory
MEM_WRITE	Write operation on up to 75% of the available memory for 30s	Memory
NETWORK_STATIC	Send and receive operations for 30s	Network
NETWORK_SEND	Send operation for 30s	Network
NETWORK_RECEIVE	Receive operation for 30s	Network
STORAGE_STATIC	Mixed read and write operations for 30s	Storage
STORAGE_READ	Read operation on storage for 30s	Storage
STORAGE_WRITE	Write operation on storage for 30s	Storage
LATENCY	Oscillating CPU load for 30s	Tools
STABILITY	Idle workload for 300s	Tools

Table B.1: Complete list of all benchmark experiments.

Appendix C

Additional Experiments

```
from Netio import Netio

granularity_values = [0, 0.001, 0.001, 0.01, 0.1, 1]
errors = []

for i in granularity_values:
    errors.append(test_granularity(i))

def test_granularity(granularity):
    PDU_L, PDU_R = setup_PDU()

    measurements_counter = 0
    error_counter = 0

    while measurements_counter < 100:
        measurements_counter++
        try:
            read_PDU(PDU_L, PDU_R)
        except:
            error_counter++
        sleep granularity;
    return error_counter
```

Listing C.1: Granularity test for Netio PowerPDUs

Listing C.1 presents the source code for the granularity benchmark experiment on the Netio PowerPDU. The execution of the benchmark experiment on the other power monitoring tools follows the same structure. The source code for all granularity benchmark experiments can be found in the `granularity.py` in the `BenchFrame` source code. The granularity experiments are executed after the benchmark experiments or can be run separately.

Appendix D

Measurement Results

Granularity	None	RAPL	Redfish	PowerPDU
0	0.0001136	0.0096288	8.049786	15.1624612
0.001	0.1077604	0.1436108	12.3605434	15.2656308
0.01	1.0111004	1.06763080	10.6432328	16.112618
0.05	5.0111864	5.07535940	15.7635772	20.0437378
0.1	10.0162304	10.0788484	19.9849842	25.2603608

Table D.1: Monitoring granularity and execution speed of the tools compared to no measurement.