TASK Quarterly 30 (2) 2026

Energy—Efficiency Support Services on the CAISE Platform

Paweł Czarnul [1,2*], **Oksana Diakun** [1,2], **Grzegorz Koszczał** [1,2], **Adam Krzywaniak** [2], **Jerzy Proficz** [1,2†], **Piotr Sokołowski** [1,2]

[1]Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Narutowicza 11/12, 80-233 Gdańsk, Poland
[2]Centre of Informatics – Tricity Academic Supercomputer and networK (CI TASK), Gdańsk University of Technology, Narutowicza 11/12, 80-233 Gdańsk, Poland

[*]pawel.czarnul@pg.edu.pl [†]jerzy.proficz@pg.edu.pl

All authors contributed equally to this work and are listed in alphabetical order by surname.

https://doi.org/10.34808/tq2026/30.2/b

Abstract

The article presents a package of services supporting energy efficiency within the CAISE platform environment, aimed at monitoring, managing, and optimizing energy consumption in cloud computing environments. The monitoring services include Monitoring Energy of Operations (MEO) and Monitoring Intensity of Operations (MIO), which enable real-time tracking of energy usage parameters and resource load. The management and administrative layer is formed by Kernel Energy Settings (KES) and Recording of Performance Optimization (RPO), which allow the definition of energy policies and the collection of data relevant for subsequent analysis and system tuning. A key component of the package is the Optimization of Workload-Energy efficiency (OWE) service, which supports the selection of configurations and execution methods for computational tasks in order to achieve a balance between performance and energy efficiency. The paper discusses the functional assumptions, supported hardware–software configurations, and implementation aspects of these services. It also outlines the requirements and challenges related to their practical deployment, potential application perspectives and use cases in large-scale computing environments, as well as selected practical results for OWE.

Keywords:

high-performance computing, energy-efficient computing

1. Introduction

The growing demand for computational power, driven by the rapid development of digital services—including artificial intelligence systems—has led to a sharp increase in energy consumption by data centers and cloud infrastructure. According to forecasts by international energy organizations [1], the share of the ICT sector in global energy demand may increase several times over the next decade, making energy efficiency one of the key technological and environmental challenges. This is particularly important in the context of AI workloads, which are characterized by high computational intensity, long task durations, and significant demand for GPU and CPU resources.

In response to these challenges, *green computing* solutions are gaining increasing importance, enabling energy consumption reduction without compromising service quality or availability. In cloud environments, which are highly scalable and flexible, energy optimization can be achieved through dedicated power management, monitoring, and intelligent resource scheduling services.

The Cloud Platform (CAISE) for developing universal intelligent services across various application domains has been designed as a cloud environment offering a set of machine learning-based services as well as services supporting energy efficiency, while also providing robust support for such workloads. These services combine mechanisms for monitoring and controlling energy consumption. When developing these solutions, high priority is placed on maintaining computational performance while significantly reducing energy usage.

As part of the work on energy-efficiency-supporting services, dedicated mechanisms are also being developed to automate energy optimization processes, including the DEPO tool [2], which enables dynamic real-time control of CPU and GPU power consumption, taking into account the characteristics of the computational workload.

The aim of this paper is to present the architecture of the CAISE platform and to describe the energy-efficiency-supporting services that are to be integrated within this platform.

The outline of the paper is as follows. In Section 2 we provide a description of related work which is followed by discussion of proposed and implemented energy-efficiency supporting services for the CAISE platform, namely: Energy Consumption Monitoring Service (MEO) – Section 3.1, Computation Intensity Monitoring Service (MIO) – Section 3.2, System Energy Configuration Service (KES) – Section 3.3, Performance-Energy Optimization Service (OWE) – Section 3.4, and Optimization Parameter Logging (RPO) – Section 4.1. We provide design and implementation details along with exemplary use cases, as well as the effects of applying the proposed mechanisms in practice. Finally, we provide conclusions and outline potential future work in Section 5.

2. Related work

Fornaciari et al. [3] present the RECIPE project, a framework targeting predictive, reliability-and-timing-aware resource management for future heterogeneous Exascale HPC systems. The core contribution of RECIPE is a hierarchical resource management architecture that combines a global resource manager (built on SLURM) with local runtime resource managers based on the BarbequeRTRM framework [4]. This design enables both proactive (prediction-based) and reactive (measurement-driven) control of resources. At the global level, coarse-grained workload allocation and cooling decisions are informed by system-wide models, while at the node level, fine-grained allocation and reconfiguration of CPUs, GPUs, FPGAs, and other accelerators are performed dynamically. A distinguishing feature of this work is its strong emphasis on predictive models. RECIPE integrates fault prediction, timing analysis, and thermal and power modeling to estimate the likelihood of meeting application deadlines under specific configurations. Reliability is treated not only as a recovery problem (e.g., checkpointing), but as a first-class scheduling constraint, particularly relevant for real-time and QoS-sensitive HPC workloads. The framework explicitly considers both short-term transient faults and long-term aging effects driven by thermal stress. Advanced monitoring infrastructure, including high-resolution power estimation and hardware performance counters, feeds the runtime managers with fine-grained telemetry required for closed-loop control. Although this paper presents a well-defined architectural concept of the RECIPE framework and identifies concrete components grounded in prior work, providing a clear description of the overall control loops, it does not include detailed scheduling or optimization algorithms, quantitative evaluation, or implementation-level details. Nevertheless, the work serves as a valuable conceptual and architectural reference for predictive runtime management in heterogeneous HPC systems.

While RECIPE focuses on architectural and runtime-level resource management in HPC systems, other works address energy efficiency at higher orchestration layers in cloud environments.

Alzahrani and Tang [5] address the problem of energy-efficient deployment of microservice-based SaaS applications in cloud data centers, focusing on minimizing the increase in energy consumption incurred by deploying new services. The authors shift

deployment responsibility to the cloud provider, enabling energy-aware placement decisions across compute servers and network devices. The deployment problem is formulated as a constrained combinatorial optimization problem, jointly considering server resource constraints, inter-microservice communication requirements, and detailed energy models for both compute servers and network switches. To solve this problem, the authors propose a Hybrid Particle Swarm Optimization (HPSO) algorithm that combines global exploration with a local optimization phase to refine energy-efficient placements. A notable strength of this work is its explicit and detailed energy modeling, capturing dynamic energy increases due to CPU workload on servers and traffic load on network devices. Experimental results show that HPSO consistently achieves additional energy savings of up to 3.68% over GA, while maintaining scalability as the number of microservices and servers increases.

Giacobbe et al. [6] propose an energy-aware brokering algorithm (eBA) aimed at improving sustainability in Community Cloud environments by minimizing carbon emissions while balancing availability and service cost. The work targets federated and cooperative cloud scenarios in which multiple medium- and small-scale cloud providers share resources, enabling energy-aware workload placement across geographically distributed data centers. The authors introduce a multi-criteria sustainability model that combines infrastructure efficiency metrics—such as Power Usage Effectiveness (PUE), Data Center Performance Per Energy (DPPE), and Carbon Dioxide Intensity of Electricity (CDIE)—into a unified sustainability impact factor. This factor is used to estimate the carbon footprint of both workload execution and virtual machine migration, explicitly accounting for availability and idle power consumption. Based on these models, the eBA algorithm evaluates competing deployment options and selects the placement that minimizes environmental impact while considering monetary cost. The proposed approach is evaluated through simulations using the J2CBROKER framework, with realistic datasets derived from commercial cloud providers and real-world energy-efficiency measurements. Experimental results demonstrate that the broker is able to consistently select deployment options with significantly lower carbon footprints, achieving a balanced trade-off between sustainability and cost across varying workloads and deployment durations.

Complementary to cloud-level deployment and brokering approaches, energy-aware service placement has also been studied in the context of service function chaining within data centers. Yang et al. [7] study the energy-aware service function placement problem for service function chaining (SFC) in data centers, considering the joint energy consumption of servers and network devices. The authors formulate the placement problem as a binary integer program that accounts for CPU and bandwidth constraints as well as detailed power models for servers, switches, and ports. To address the problem's computational complexity, they propose the Merge-RD heuristic, which introduces a Relation Degree metric to capture traffic dependency and ordering among service functions. By co-locating highly related functions and consolidating network flows, the approach reduces both energy consumption and transmission delay. Simulation results on fat-tree data center topologies demonstrate improved energy efficiency compared to existing service placement algorithms while maintaining network performance guarantees.

The reviewed works demonstrate that energy efficiency has been addressed at multiple layers of the computing stack, from predictive, fine-grained runtime management in HPC systems to energy-aware deployment, brokering, and service placement in cloud and data center environments. While frameworks such as RECIPE emphasize hierarchical control and predictive models, they primarily target tightly coupled HPC systems and do not address energy efficiency as a cloud-level service. Conversely, cloud-oriented approaches focus on optimization at specific orchestration layers, often without tight integration with low-level runtime monitoring and control.

In contrast, the CAISE platform targets energy efficiency as a first-class capability within an IaaS cloud offering, integrating energy-efficiency support services alongside AI services. By bridging infrastructure-level monitoring and control with cloud-level orchestration, CAISE complements existing approaches and addresses the gap between low-level energy-aware resource management and service-oriented cloud platforms.

3. CAISE platform architecture and energy-efficiency supporting services

Our platform has a quite typical layered architecture for cloud environments. The base layer is the infrastructure layer, which consists of physical servers equipped with CPUs and GPUs, and also networking and storage devices. These layers are built with Ubuntu 24.04 LTS as the operating system. On top of the infrastructure layer, there is the virtualization layer, which is based on OpenStack. The OpenStack platform runs inside containers managed with Docker and Ansible. At this level, the monitoring of every node is performed using Telegraf, which is of course placed inside the Docker container. Atop the virtualization layer, there is the

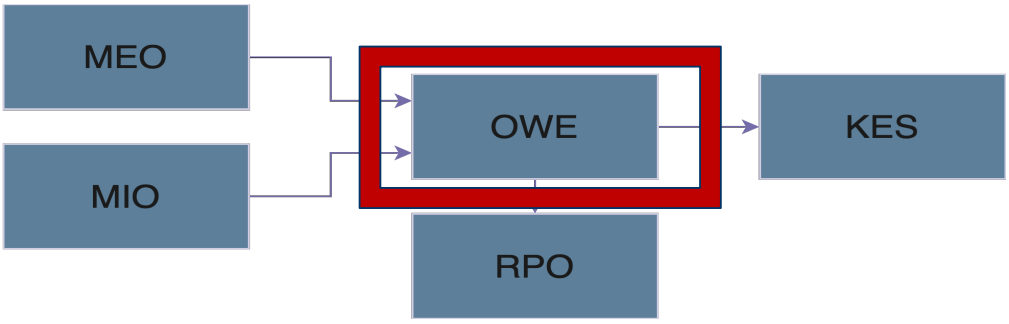


Figure 1: Overview of CAISE energy-supporting services.

orchestration layer, which is based on Kubernetes. All the data collected by Telegraf and the other monitoring services that run inside Kubernetes are stored inside the Mimir time-series database and are presented with the Grafana dashboard. From the energy efficiency point of view, the most important devices are CPUs and GPUs, and these are different in terms of monitoring. The CPUs can be monitored all the time, nevertheless this works for a host or VM, but this is not the same in the case of GPUs. The GPUs are managed with GPU passthrough technology, which allows physical GPUs to be assigned directly to the virtual machines running inside the Kubernetes cluster. This solution has some limitations, as the device cannot be shared between the host and the virtual machine, so when the GPU is assigned to the VM, it is not available for the host system even for monitoring purposes. Therefore, we decided to monitor the GPUs only when they are assigned to the VM, from the Kubernetes, but CPUs are monitored all the time from the host system.

The energy-supporting services within the CAISE platform are ultimately aimed at providing workload agnostic optimization of the cloud environment operation. These services, described in detail in the following subsections, have been designed to complement each other in this context. Dependencies between these services are outlined in Figure 1 with OWE marked as the key – optimization – service. Specifically, the monitoring services Energy Consumption Monitoring Service (MEO) and Computation Intensity Monitoring Service (MIO) provide a means of monitoring energy/power and metrics related to computing devices' execution engines for indirect workload progress monitoring. These are used directly by the aforementioned key service – Performance-Energy Optimization Service (OWE). Configuration of the OWE service and the whole energy-supporting environment can be controlled through the System Energy Configuration Service (KES) and its interface. Finally, Optimization Parameter Logging (RPO) supports recording important events and data for subsequent analysis and recall, if needed.

Within the CAISE platform, the proposed energy-efficiency services constitute a horizontal support layer available to all workloads deployed in the cloud environment, including AI workloads. While CAISE provides the underlying infrastructure, virtualization, and orchestration mechanisms required for service deployment and execution, the presented services extend the platform with advanced capabilities for monitoring, control, optimization, and analysis of energy consumption. As a result, energy efficiency becomes an integral platform capability rather than an application-specific feature implemented independently by individual users.

From the broader CAISE perspective, these services enhance the platform by enabling sustainability-aware operation of cloud resources without requiring modifications to user applications. The proposed approach allows optimization mechanisms to operate transparently in the background, reducing energy consumption while preserving acceptable performance levels. This is particularly important for modern AI and high-performance workloads, where energy costs and environmental impact are becoming increasingly important deployment criteria. Consequently, the presented services contribute directly to the CAISE vision of delivering scalable, intelligent, and sustainable cloud services.

3.1. MEO

To monitor the computational environment in terms of energy consumption, the Monitoring Energy of Operations Service (MEO) was created. The service simultaneously uses low-level mechanisms for measuring the power consumption and energy usage of general-purpose processors (CPU) as well as accelerators such as graphics cards (GPU).

To observe the desired parameters, Telegraf [8] was used—a tool that offers a range of necessary interfaces. To meet the service requirements, the input plugins `intel_powerstat` and `nvidia-smi` were selected. The `intel_powerstat` plugin uses the RAPL (Running Average Power Limit) mechanism, while the nvidia-smi (NVIDIA System Management Interface) plugin is a tool and command-line interface provided by NVIDIA

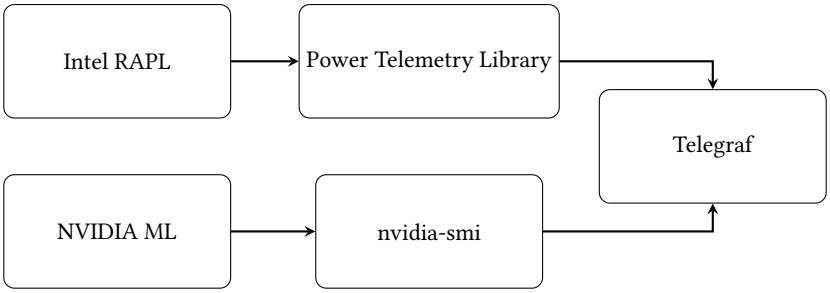


Figure 2: Block diagram representing the solution

that enables monitoring and management of graphics cards. For both plugins, in the context of this service, the fields related to power consumption are of particular interest, namely `current_power_ consumption` and `power_draw`.

In the context of reading parameters from the main processors (CPU), this service uses a software library written in Go. The data source is the Linux kernel interface, which provides an energy consumption counter expressed in microjoules. The conversion to power is performed inside the Power Telemetry Library [9], which measures the power draw over time intervals expressed in seconds. In the implemented solution, the exact interval value depends on the polling frequency of the library, which in this case is defined by Telegraf. As a result, it is relatively easy to convert the calculated power back into energy, eliminating the need to transmit and store these parameters separately, which saves both storage space and bandwidth.

Monitoring for graphics cards is implemented differently, even though a Go library is also available for this purpose [10]. The data source is the NVIDIA Management Library, but the readings are obtained through the `nvidia-smi` tool, which, in the case of A100 cards, reports power for a 1-second interval [11]. To obtain complete information on total energy consumption over time, regular readings are required. The data from the tool are returned in XML format, which may potentially require adjustments in the future if the XML schema changes and the corresponding Telegraf plugin is not updated accordingly. This issue occurs, for example, with `nvidia-smi` version `535.154.05` and `Telegraf 1.27.1`. Updating Telegraf to version `1.32.1` resolves the problem. An alternative solution would be to use a DTD that can be added to the returned XML by specifying the —dtd option, but such functionality has not currently been implemented in the plugin. The block diagram of the solution is shown in Figure 2.

The data collected from monitoring are stored in the internal Mimir database [12], which is accessible to the administrator. This database was developed by GrafanaLabs. Data representation is enabled through the Grafana tool [13], also developed by the same company. Both solutions are available as open source. Grafana allows for data analysis and interactive visualization.

3.2. MIO

Every device has its own computational intensity metrics. For example for CPUs, it can be instructions per cycle (IPC) or floating point operations per second (FLOPS). For GPUs, the problem is more complex, as different types of operations can be performed (FP16, FP32, FP64, Tensor Cores, etc.). Therefore, monitoring of the computational intensity of GPUs is usually only approximated by measuring the utilization of streaming multiprocessors (SM utilization) or memory controllers (memory utilization). So, in our MIO service, we decided to use SM utilization, engine active and FP16/32/64 pipe active parameters as the base for the computational intensity metric. The implementation of the MIO service for CPUs is similar to the MEO service, described in Section 3.1. We also use Telegraf with Intel PMU plugins, a data collector and Prometheus/Mimir as a time-series database. For GPUs, we use DCGM (Data Center GPU Manager) as a data source, so the solution is a little bit different. We install it on the Kubernetes node inside the VM, so we need an additional Prometheus Agent to scrape DCGM metrics and send them to the Mimir database. All collected data are also visualized in the Grafana tool.

3.3. KES

KES is a service that helps administrators in management of power-capping of devices in both bare metal and Kubernetes deployment scenarios as well as in changing the configuration of the OWE service, such as:

- optimization algorithm,
- target metric,
- how often the re-tuning process should take place,
- application tuning window.

This is achieved by REST API, written in Python with FastAPI as a web framework.

Power-capping related functionalities are provided by reliable Python modules, such as `amdsmi` for AMD CPUs, `py-cpuinfo` for Intel CPUs and `nvidia-ml -py` for NVIDIA GPUs. The OWE configuration part works both on bare metal and in Kubernetes.

On bare metal, the application receives and validates requests and modifies the configuration file, located on `/etc/depo-daemon/config.yaml` filepath, using `yaml.safe_load()`.

In a Kubernetes cluster, the application uses the `Kubernetes` module and its methods to safely read and patch the `ConfigMap` resource – an OWE configuration object, that resides inside the cluster's space.

At the startup, whether in the Docker container on bare metal or in Pod in Kubernetes, the application performs two checks which work for the entire lifetime of the service, saving performance and ensuring stability.

The first check involves device probing – the application checks:

1. CPU power-capping capabilities. This works only on bare metal, due to the fact that CPU passthrough is impossible on virtual machines where Kubernetes operates.
   (a) If the condition is met, then the application resolves a CPU manufacturer and possible NUMA cores to power-cap.
2. GPU power-capping capabilities. This scenario works both on bare metal and in Kubernetes, if a device is present on the host machine or the virtual machine.
   (a) If the condition is met, the application resolves a GPU manufacturer and possible CUDA devices to power-cap.

Currently, the following device product lines are supported: Intel Xeon and Core CPUs, AMD EPYC CPUs and NVIDIA Data Center GPUs.

This approach ensures that a user cannot access devices that do not meet conditions for power-capping, acting as a validation and security.

The second check involves a deployment platform – the application checks:

1. If the application operates on a bare metal environment.
   (a) If the condition is met, then a Docker container with service accesses an OWE configuration file located on a host machine and can apply patches to it.
2. If the application operates in the Kubernetes cluster.
   (a) If the condition is met, then Pod with services accesses ConfigMap the object within the cluster's space and can apply patches to it.

As a result, the application requires less attention from users and makes deployment easier, ensuring robustness.

Figure 3 shows all available endpoints, related to both power-capping and configmap edition capabilities.

Figure 3: Overview of KES endpoints.

Exemplary usage of KES endpoints, shown by using Postman API testing, are presented in Figures 4a-5b.

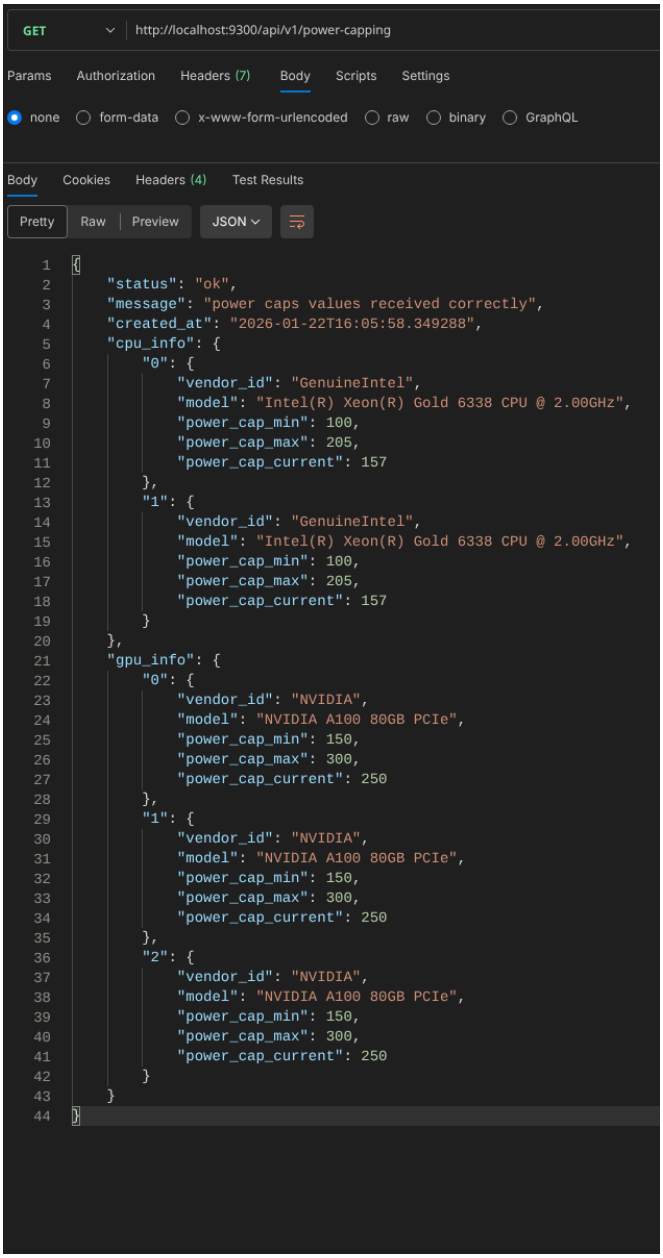
3.4. OWE Service: From Job-Centric Optimization to a Daemon-Based Architecture

3.4.1 Motivation and Architectural Shift

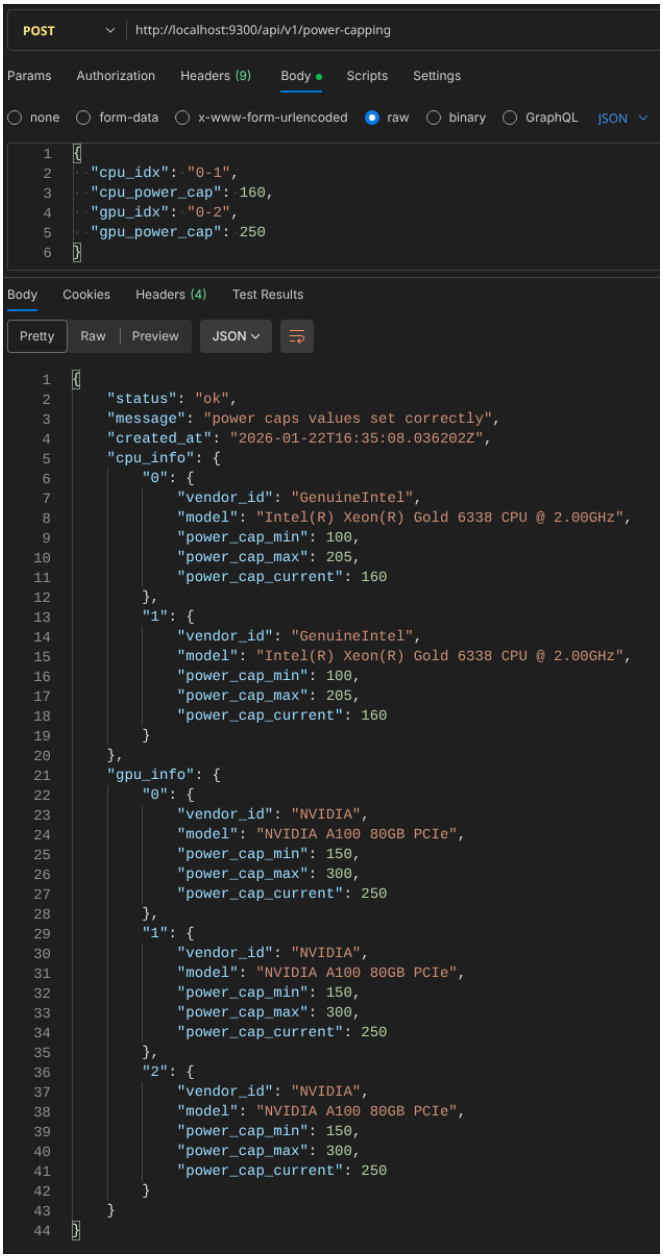
The original DEPO tool was designed as a job-centric optimizer, operating on a single workload instance and dynamically adjusting power limits during its execution in order to optimize selected energy–performance metrics, such as total energy consumption, Energy–Delay Product (EDP), or Energy–Delay Sum (EDS) [2, 14]. While this approach proved effective for standalone HPC and accelerator workloads, it inherently couples the optimization process to the life cycle of a specific job.

As a next step in the evolution of DEPO, we introduce OWE – a service-oriented extension that transforms DEPO into a daemon-based architecture. The main objective of OWE is to decouple energy–performance optimization from individual job execution and enable continuous, system-level optimization driven by real-time power measurements and hardware performance indicators.

This architectural shift allows DEPO-based optimization to operate persistently in the background, independently of workload lifetimes, and across heterogeneous compute devices, while still relying on dynamic power capping as the primary control mechanism.



(a) Retrieval of devices power-caps via GET method.



(b) Setting devices power-caps via POST method.

Figure 4: Power-capping API usage examples.

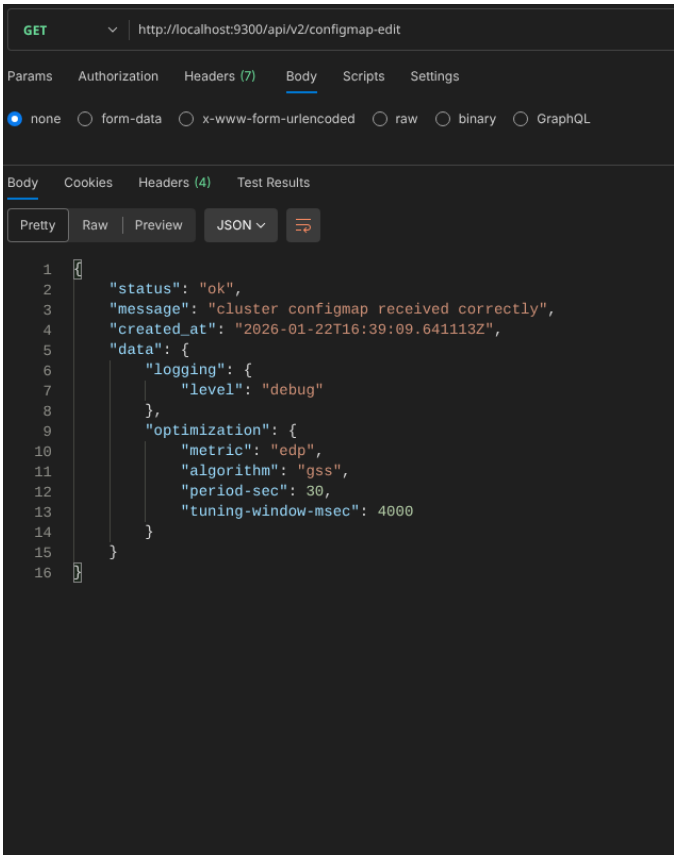
3.4.2 Daemon-Based Design

OWE is implemented as a new system-level application, `depo-d`, which runs as a persistent daemon. In contrast to the original DEPO workflow, where the optimizer is launched together with the optimized application, `depo-d` continuously monitors the system and periodically performs the following steps: (i) acquisition of current power consumption, (ii) collection of hardware performance counters characterizing computational efficiency, (iii) evaluation of the selected optimization objective, and (iv) adjustment of device power limits. The periodic mode of DEPO and its increased stability versus one-time tuning is demonstrated in the paper [15].

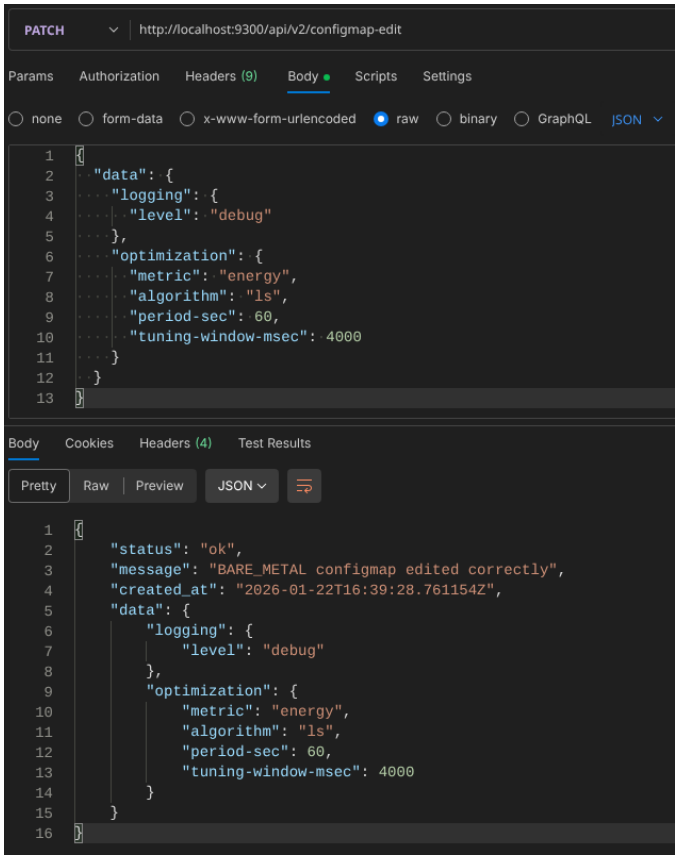
This design enables OWE to react to changes in workload characteristics without requiring explicit knowledge of application structure or tight coupling with runtime environments.

3.4.3 CPU Power and Performance Monitoring

On Intel CPUs, OWE reuses well-established interfaces already employed in DEPO: power consumption is measured using Running Average Power Limit (RAPL), power limits are enforced via RAPL, and hardware performance monitoring is performed using Intel Performance Counter Monitor (PCM). In particular, instruction count readings are used as a low-level indicator of computational progress and efficiency, enabling correlation of energy consumption with useful work performed by the processor.



(a) **Retrieval of DEPO config map via GET method.**



(b) **Changing DEPO config map via PATCH method.**

Figure 5: DEPO ConfigMap editing API usage examples.

3.4.4 GPU Power and Performance Monitoring

For NVIDIA GPUs, OWE relies on NVIDIA Management Library (NVML) for both power consumption measurement and power limit enforcement. Hardware performance monitoring is implemented using NVIDIA Data Center GPU Manager (DCGM).

Specifically, OWE uses DCGM performance counters such as DCGM_FI_PROF_ PIPE_FP32_ACTIVE, as well as the corresponding counters for FP16 and FP64 operations. These counters report the utilization of floating-point execution pipelines and provide an indirect measure of computational efficiency.

3.4.5 DEPO-based Cloud Energy Optimizer (CEO) in a Kubernetes Environment

The Cloud Energy Optimizer (CEO) is implemented as a daemon-based extension of DEPO and reuses the original dynamic energy–performance optimization logic without modification. In particular, the core optimization routine invoked by CEO is directly inherited from DEPO and executed through the method runDynamicEnergyPerformanceOptimization(), ensuring behavioral equivalence between job-centric and daemon-based optimization modes.

CEO operates as a long-running system service that continuously monitors the utilization state of the GPU and dynamically applies power-cap tuning only when active workloads are detected. GPU activity is identified using NVIDIA Management Library (NVML) by querying the presence of running compute processes. When the GPU is idle, CEO restores default power limits, avoiding unnecessary interference with the system.

Optimization parameters, such as the target metric (e.g., energy or Energy–Delay Product), search algorithm (linear search or golden section search), tuning window, and optimization period, are provided via an external configuration file. This design enables dynamic reconfiguration of optimization behavior without restarting the service and supports deployment in managed environments.

In a Kubernetes-based infrastructure, CEO is deployed as a node-level service using a DaemonSet, ensuring that exactly one instance runs on each compute node equipped with a GPU. The CEO container is executed with elevated privileges and access to host device interfaces, allowing it to interact directly with physical hardware through NVML and to enforce GPU power limits at the node level. As a result, CEO operates independently of individual containers and applies optimization decisions globally to all workloads executing on the corresponding GPU.

This deployment model preserves the original

Table 1: Comparison of execution time, energy consumption, average power, and EDP with and without power capping on NVIDIA H100.

Scenario	Time [hh:mm:ss]	Energy [MJ]	Avg. Power [W]	EDP [J·s]
No power cap	24 : 04 : 09	43.827	505.80	3.7976×10^{12}
With power cap	25 : 56 : 39	35.908	384.46	3.3538×10^{12}
Change [%]	+7.79	−18.07	−23.99	−11.69

DEPO optimization semantics while enabling continuous, system-level energy–performance optimization for dynamically scheduled and short-lived container workloads. By decoupling power-cap tuning from application lifetimes and container boundaries, CEO extends the DEPO approach to cloud-native and hybrid HPC environments without requiring application instrumentation or scheduler-level modifications.

Replacing CUPTI with DCGM in a Daemon Architecture In earlier GPU-oriented versions of DEPO, application progress was estimated using a proxy module based on the CUDA Profiling Tools Interface (CUPTI), which counted kernel invocations during execution [14]. This approach provided a direct notion of workload progress but required CUPTI to instrument a specific target process by injecting atomic instructions into its execution.

Such a mechanism is inherently incompatible with a daemon-based architecture. In the OWE model, the optimizer operates independently of any particular application process and must handle dynamically changing and previously unknown workloads. Since CUPTI-based profiling requires explicit attachment to a concrete process, it does not scale to scenarios where no single, long-lived target process exists or where workloads change over time.

For this reason, CUPTI-based kernel counting was replaced with DCGM-based performance monitoring. Although DCGM counters do not directly represent application progress, comparative experiments demonstrated sufficient convergence between CUPTI-based and DCGM-based optimization outcomes. This trade-off enables a significantly simpler and more robust monitoring stack suitable for continuous daemon operation.

3.4.6 Optimization Model and Scope

OWE preserves the core optimization philosophy of DEPO: dynamic adjustment of power limits based on runtime observations in order to optimize selected energy–performance metrics. While earlier DEPO implementations focused on per-job optimization, OWE generalizes this concept to continuous system-level operation by evaluating instantaneous power consumption and hardware efficiency indicators.

As a result, OWE enables adaptive energy–performance optimization for long-running, mixed, or time-varying workloads without requiring prior knowledge of application behavior.

4. Preliminary results

In this section, results from two initial experimental runs are presented. Both instances of experiments concern the environment of compute006, a host equipped with AMD EPYC 9334 CPU and NVIDIA H100 GPU. First, the case with evaluation conducted on the GPU is presented, followed by the CPU case. To emulate a realistic and time-varying usage scenario, the workloads were executed as part of a long-running experiment. During this period, the workloads were repeatedly launched in multiple iterations, interleaved with idle phases of varying duration (ranging from a few seconds to several tens of seconds). Such workload composition reflects practical deployment conditions in shared or service-oriented environments, where computationally intensive workloads coexist with intermittent inactivity and irregular execution patterns. The resulting variability in power consumption and utilization makes this scenario particularly suitable for evaluating the effectiveness of daemon-based, runtime energy-performance optimization mechanisms.

Firstly, a long-running experiment of approximately 24 hours was conducted on the GPU under a variable workload. The experimental workload was based on *XRA-Diff*, a diffusion-based refinement method for backprojected coronary angiography X-ray images. The workload was executed in a containerized environment using Docker and deployed on a GPU-enabled system. Each execution consisted of a single inference run of the XRA-Diff pipeline, configured with a fixed number of diffusion steps (`STEPS=1000`), ensuring consistent computational characteristics across runs.

The experimental workload was executed on a single GPU, while the remaining devices remained idle. GPU power monitoring and power limit enforcement were performed using NVML, and hardware performance metrics were collected via DCGM. The CPU in this scenario was primarily used for orchestration and data handling, while

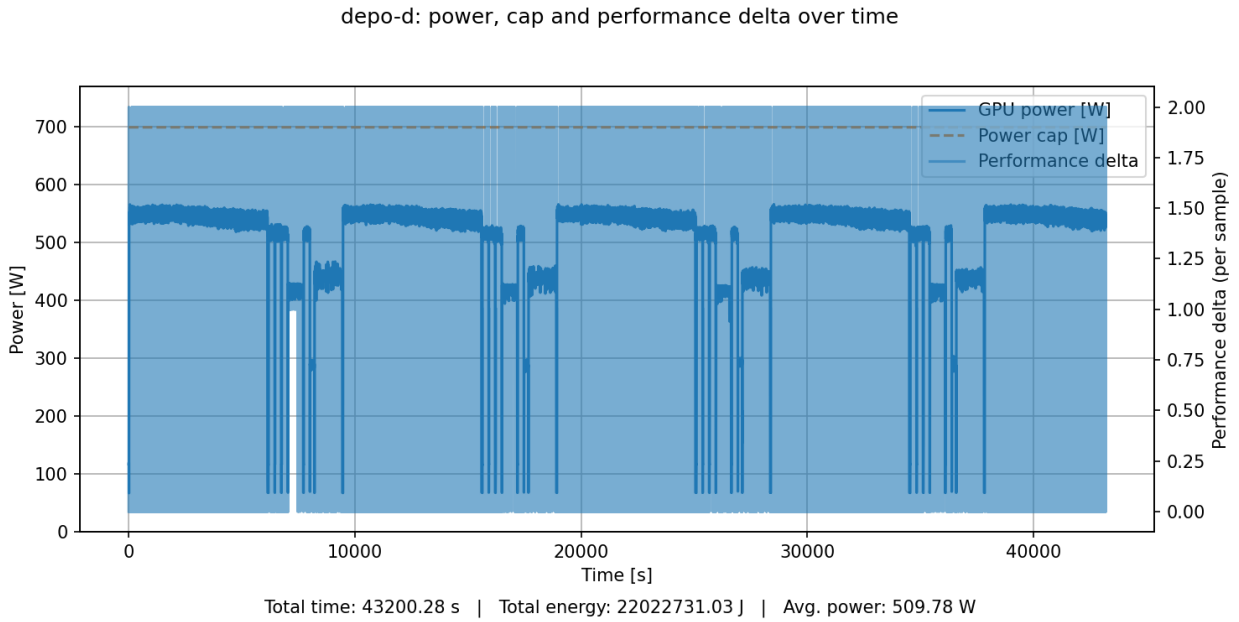


Figure 6: Power consumption profile during the first 12 hours of execution without power capping on NVIDIA H100.

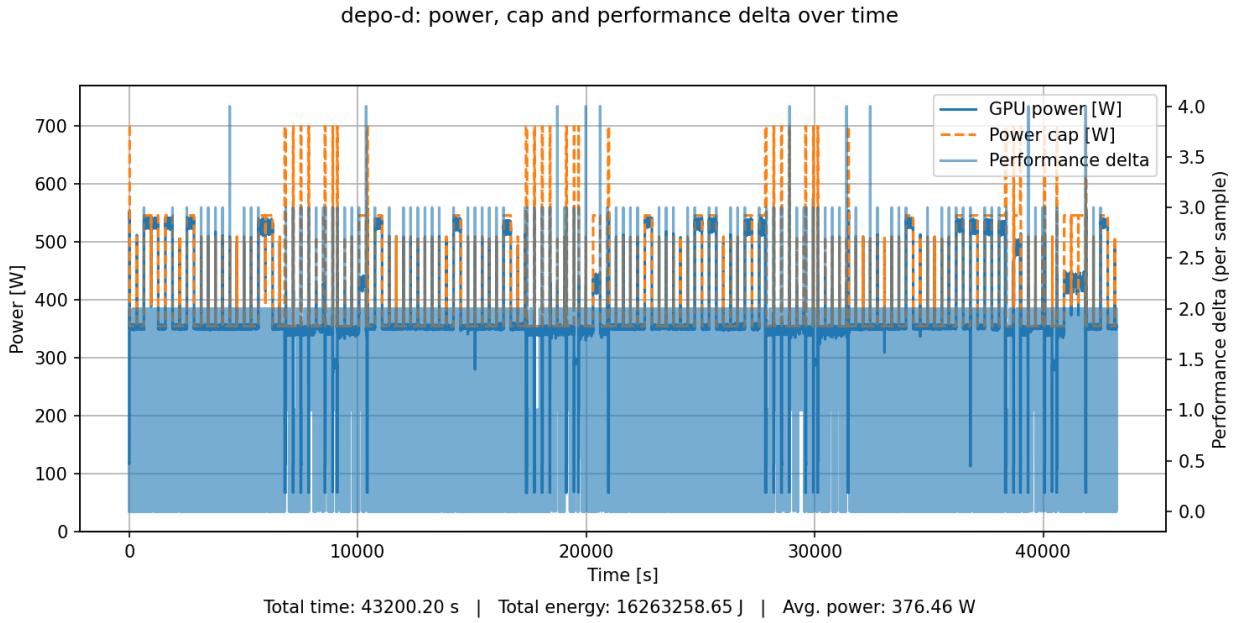


Figure 7: Power consumption profile during the first 12 hours of execution with `depo-d` optimizing the Energy metric on NVIDIA H100.

the computational workload was GPU-bound.

Figure 6 shows the behavior of GPU power consumption over 12 hours on an NVIDIA H100 without active power capping. The GPU power remains mostly in a high and stable operating region of approximately 540–560 W throughout the majority of the execution. Periodic sharp drops in power consumption correspond to the completion of repetitive workload iterations and the short intervals associated with relaunching the workload.

Figure 7 illustrates the GPU power consumption behavior with `depo-d` actively optimizing the energy metric. In contrast to the non-tuned execution, the power cap is dynamically adjusted throughout runtime, resulting in frequent transitions between different GPU power states. The GPU power consumption closely follows the selected power cap values, demonstrating the effectiveness of runtime power control. Most of the execution is performed at significantly lower power levels of approximately 350–380 W, reducing the average GPU power consumption to 376.46 W. Occasional transitions to higher power levels indicate exploratory or performance-recovery phases of the tuning algorithm. Periodic sharp drops in power consumption correspond to workload completion and relaunch intervals.

The results indicate the potential for significant energy savings exceeding 18%, while incurring a performance penalty below 8%, expressed as an increase in

Table 2: Comparison of execution time, energy consumption, average power, and EDP with and without power capping on AMD EPYC 9334 CPU.

Scenario	Time [hh:mm:ss]	Energy [MJ]	Avg. Power [W]	EDP [J·s]
No power cap	05 : 11 : 10	6.784	363.35	126.64×10^9
With power cap	06 : 34 : 51	6.646	280.51	157.46×10^9
Change [%]	+26.91	−2.04	−22.80	+24.33

total execution time. Table 1 summarizes representative quantitative results obtained during one of the initial experiments.

The second experiments concern workloads consisting of mini-benchmarks utilizing OpenMP. One of them –./heat– solves the diffusion equation for heat propagation in a 2D grid and takes arguments such as grid size and the number of iterations. The other –./fft– computes the Fast Fourier transform for a vector size and the number of iterations. By selecting different values for the task arguments and interleaving tasks with idle periods of varying durations, it was possible to adjust the total duration of the experiments to fit the allotted time.

Table 2 presents a comparison of the execution time, total energy consumption, average power, and Energy-Delay Product (EDP) for the AMD EPYC 9334 CPU with and without power capping. Applying the power cap reduced the average CPU power consumption from 363.35 W to 280.51 W, corresponding to a 22.80% decrease. This reduction also led to a slight decrease in total energy consumption from 6.784 MJ to 6.646 MJ, improving energy efficiency by 2.04%. However, the lower power limit significantly increased the execution time from $\approx 5h$ to $\approx 6.5h$, representing a 26.91% performance degradation. As a consequence, the EDP increased by 24.33%, from 126.64×10^9 J·s to 157.46×10^9 J·s, indicating that the reduction in power and energy consumption was outweighed by the longer execution time. These results suggest that aggressive CPU power capping on the AMD EPYC 9334 improves power efficiency but negatively impacts overall performance efficiency.

Figure 8 illustrates the CPU power consumption behavior on the AMD EPYC 9334 without active power capping. The CPU operates predominantly near the maximum observed power region of approximately 450-480 W during workload execution phases, indicating sustained high computational utilization. Periodic drops in power consumption to significantly lower levels correspond to workload completion and transient idle intervals before subsequent workload relaunches. The power profile exhibits a repetitive execution pattern with relatively stable high-power operating regions separated by short low-power phases. Since no runtime power management is applied, the system naturally maintains high power consumption throughout the execution, resulting in an average power of 363.35 W and a total energy consumption of 6.784 MJ over the measured interval.

Figure 9 presents the CPU power consumption profile for the AMD EPYC 9334 with `depo-d` actively optimizing the Energy metric through dynamic power capping. In contrast to the non-capped execution, both the power cap and the measured CPU power vary significantly throughout runtime, demonstrating adaptive runtime power management behavior. The tuning mechanism frequently transitions between different power limits, causing the CPU to operate across a broad range of power states, typically between approximately 150 W and 400 W. Compared to the baseline execution, the average CPU power is substantially reduced to 280.51 W, contributing to a lower total energy consumption of 6.646 MJ. However, the increased variability and prolonged operation at lower power states also lead to longer execution time, as reflected by the higher EDP value. Similar to the non-capped scenario, periodic sharp power drops correspond to workload completion and relaunch intervals.

The preliminary experiments demonstrate the feasibility of the proposed daemon-based runtime optimization approach for both GPU- and CPU-oriented workloads. The results show that dynamic power management can effectively reduce power consumption while adapting to time-varying execution patterns consisting of alternating computation and idle phases. In the GPU case, the optimization process achieved favorable energy-performance trade-offs, indicating strong potential for energy-aware tuning of compute-intensive workloads. In contrast, the CPU experiments revealed that although power reduction was achievable, the resulting performance degradation limited the overall optimization effectiveness. These observations suggest that the behavior and optimization potential are strongly dependent on workload characteristics and hardware architecture. Overall, the obtained results confirm the applicability of the proposed approach and motivate further evaluation using additional workloads, metrics, and optimization strategies.

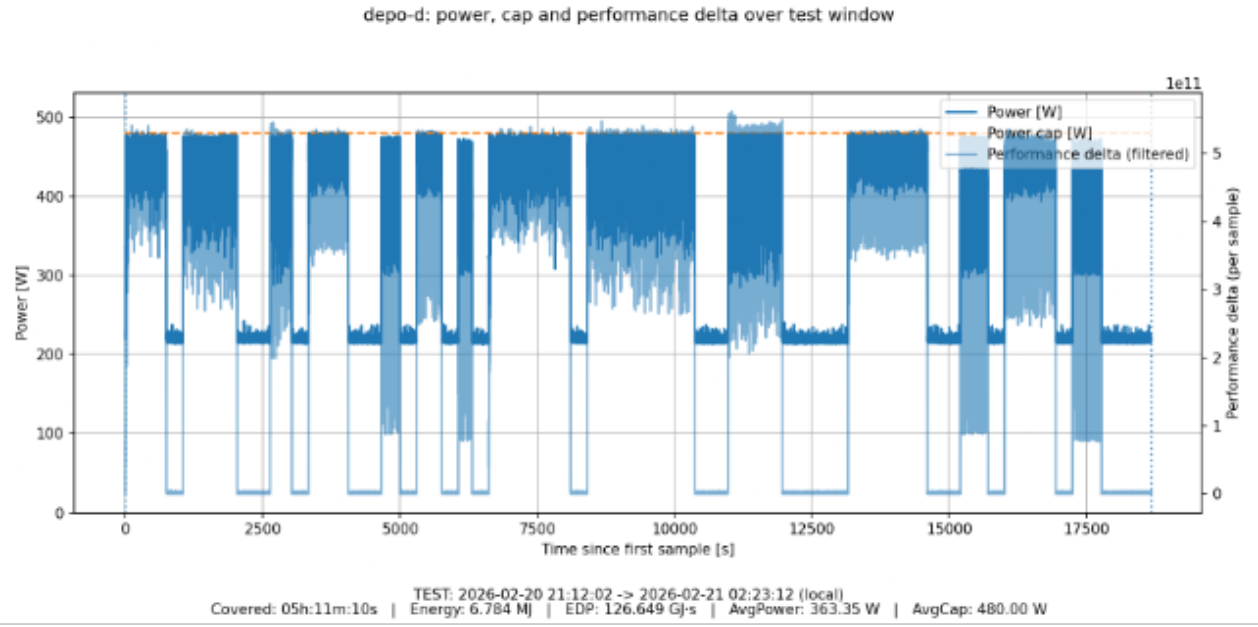


Figure 8: Power consumption profile during the first 12 hours of execution without power capping on AMD EPYC 9334.

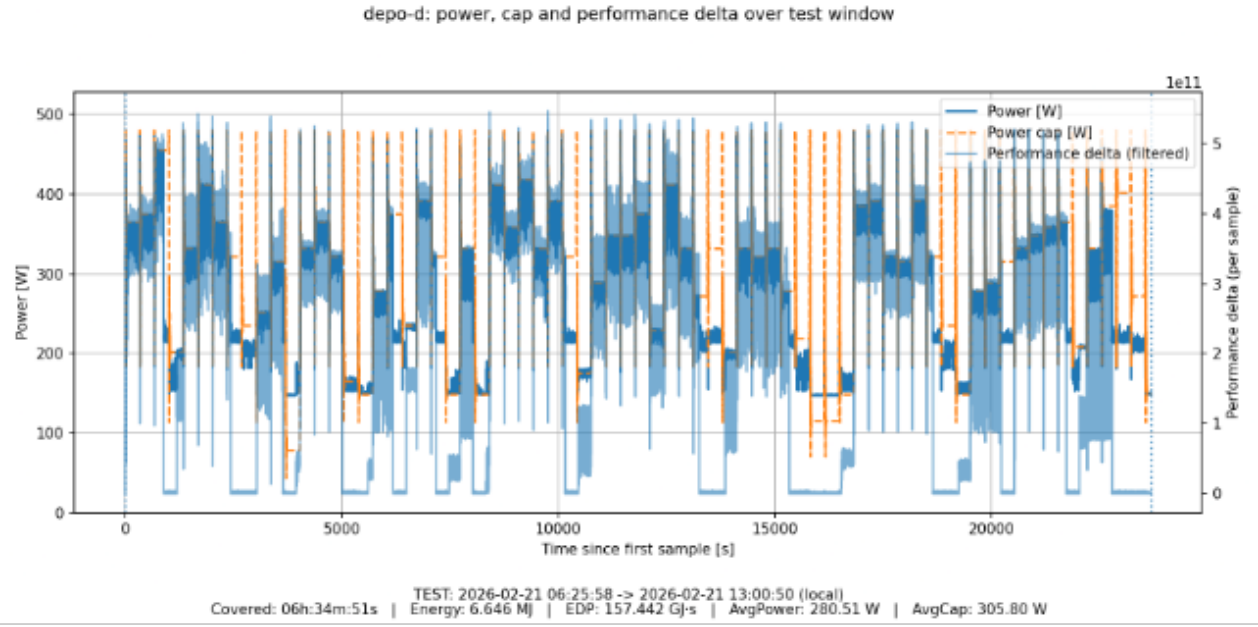


Figure 9: Power consumption profile during the first 12 hours of execution with `depo-d` optimizing the Energy metric on AMD EPYC 9334.

4.1. RPO

The overall goal of the proposed monitoring approach is to continuously record selected low-level hardware metrics, store them over long time horizons, and analyze trends and trade-offs, as well as to visualize and compare experiments or application-level effects. To achieve this goal, and considering specific hardware characteristics and constraints, a dedicated monitoring stack is required.

In this work, a monitoring stack composed of NVML, DCGM, Telegraf, Prometheus, Mimir, and Grafana is presented. This stack enables continuous, fine-grained recording of power, utilization, and performance-related metrics, while supporting long-term storage and experiment-level comparison.

4.1.1 CPU Telemetry and Power Measurements

CPU power and energy telemetry is collected using Telegraf, which provides access to low-level hardware interfaces through dedicated input plugins. In particular, CPU energy measurements are obtained via the `intel_powerstat` plugin, which relies on Intel's RAPL (Running Average Power Limit) mechanism.

RAPL exposes energy consumption counters for CPU power domains, such as the processor package and, where available, DRAM. These counters are maintained

by the Linux kernel and updated at millisecond-level resolution, enabling fine-grained analysis of CPU energy usage. Instantaneous power is derived from energy differences over successive sampling intervals, with the effective temporal resolution defined by Telegraf's polling frequency.

4.1.2 GPU Telemetry Collection

Low-level GPU telemetry is obtained using NVIDIA's management interfaces. NVML provides direct access to instantaneous device measurements, such as power consumption and power limits. Building on NVML, DCGM exposes a richer set of performance counters, including streaming multiprocessor (SM) activity, tensor core utilization, floating-point operation activity (FP16/FP32/ FP64), and memory throughput.

DCGM metrics are collected via a Prometheus-compatible exporter, allowing them to be sampled periodically and integrated into a unified time-series database alongside CPU and system-level metrics.

The CPU telemetry obtained via Intel RAPL complements GPU power measurements collected using NVIDIA management interfaces, enabling joint analysis of CPU and GPU energy consumption. Together, these measurements capture the primary dynamic power components of a compute node and provide a practical basis for energy- and power-aware optimization studies in heterogeneous CPU–GPU systems.

4.1.3 Time-Series Storage and Labeling

Metrics are scraped and stored using Prometheus, which provides a multi- dimensional time-series data model. In addition to raw measurements, optimization and experiment parameters are encoded as metric labels. Encoding optimization parameters as labels enables direct aggregation and comparison across different experimental configurations using PromQL queries, without requiring post-processing of external logs.

4.1.4 Long-Term Retention and Scalability

To support long-term analysis and reproducibility across multiple experimental campaigns, Prometheus metrics are forwarded via the `remote_write` mechanism to Mimir, a horizontally scalable backend compatible with the Prometheus query model. This approach overcomes the retention and scalability limitations of standalone Prometheus deployments, enabling persistent storage of metrics across nodes and experimental runs.

4.1.5 Visualization and Analysis

Grafana is an open-source platform for querying, visualizing, and exploring data from multiple sources. It enables the construction of interactive dashboards that present metrics in a unified web-based interface, facilitating the analysis of system behavior in relation to selected optimization parameters. Grafana connects directly to supported data sources without requiring data migration, allowing insights to be shared efficiently across experiments and users [13].

4.1.6 Rationale for Stack selection

The selected monitoring stack combines low-overhead hardware access (NVML), semantically rich GPU performance counters (DCGM), flexible metric collection and preprocessing (Telegraf), multi-dimensional time-series storage (Prometheus), scalable long-term retention (Mi-mir), and expressive visualization capabilities (Grafana). Together, these components provide a reproducible and extensible framework for recording and analyzing optimization parameters in power- and energy-aware GPU computing experiments.

Using the chosen stack, a monitoring infrastructure is obtained that integrates seamlessly with existing services such as MEO, MIO, and OWE, enabling unified data collection and analysis across the broader system ecosystem.

5. Conclusion and future work

The paper presents a CAISE-centric, platform-integrated collection of energy-related services. These services cover the most important areas of energy-aware computing: monitoring (MIO, MEO), control (KES), optimization (OWE), and logging (RPO). As such, they constitute a significant element of a modern service-engineering approach, in which energy efficiency is treated as a first-class, continuously available platform capability rather than an application-specific optimization. From the practical point of view, concerning performance-energy optimization of actual workloads, of key importance is our demonstration of a working OWE that allowed an energy reduction of 18.07% to be achieved at the cost of only 7.79% increased execution time.

The resulting foundation of services provides a solid basis for further research and development in energy-aware AI computing, both within the CAISE project and beyond. In particular, it enables the construction of higher-level optimization mechanisms, adaptive AI services, and sustainability-oriented resource management strategies that can operate across heterogeneous hardware platforms and cloud-native execution

environments.

Future work can be grouped into the following three main research directions:

1. Advanced optimization strategies, including faster and more robust tuning procedures, improved power-cap selection mechanisms, and the exploration of multi-objective optimization approaches that balance performance, energy consumption, and efficiency metrics.
2. Platform- and scheduler-level integration, in which the OWE service will be more tightly coupled with cloud orchestration mechanisms. This includes extending optimization beyond individual devices toward coordinated CPU-GPU tuning and enabling interaction with scheduling and placement decisions in cloud-native environments.
3. Broader metrics and sustainability context, incorporating carbon-related indicators alongside traditional energy and performance metrics. This direction also includes adapting optimization objectives to dynamic external factors, such as electricity market prices and the carbon intensity of energy sources.

Acknowledgments

The research was supported [in part] by project "Cloud Artificial Intelligence Service Engineering (CAISE) platform to create universal and smart services for various application areas", No. KPOD.05.10-IW.10-0005/24, as part of the European IPCEI-CIS program, financed by NRRP (National Recovery and Resilience Plan) funds.

References

[1] International Energy Agency, "Energy demand from ai," 2024.

[2] A. Krzywaniak, P. Czarnul, and J. Proficz, "Depo: A dynamic energy-performance optimizer tool for automatic power capping for energy efficient high-performance computing," *Software: Practice and Experience*, vol. 52, no. 12, pp. 2598–2634, 2022.

[3] G. Agosta, W. Fornaciari, D. Atienza, R. Canal, A. Cilardo, J. F. Cardo, C. H. Luz, M. Kulczewski, G. Massari, R. T. Gavilá, *et al.*, "The recipe approach to challenges in deeply heterogeneous high performance systems," *Microprocessors and Microsystems*, vol. 77, p. 103185, 2020.

[4] P. Bellasi, G. Massari, and W. Fornaciari, "Effective runtime resource management using linux control groups with the barbequertrm framework," *ACM Trans. Embed. Comput. Syst.*, vol. 14, Mar. 2015.

[5] A. Alzahrani and M. Tang, "Energy-aware microservice-based saas deployment in in a cloud data center using hybrid particle swarm optimization," *IEEE Access*, 2024.

[6] M. Giacobbe, M. Scarpa, R. Di Pietro, and A. Puliafito, "An energy-aware brokering algorithm to improve sustainability in community cloud.," in *SMARTGREENS*, pp. 166–173, 2017.

[7] K. Yang, H. Zhang, and P. Hong, "Energy-aware service function placement for service function chaining in data centers," in *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2016.

[8] influxdata, "Telegraf (oss), time-series platform."

[9] "Power Telemetry Library."

[10] "Go Bindings for the NVIDIA Management Library (NVML)."

[11] NVIDIA Corporation, "NVML API Reference Guide: nvmlDeviceGetPowerUsage."

[12] Grafana Labs, "Mimir."

[13] Grafana Labs, "Grafana."

[14] A. Krzywaniak, P. Czarnul, and J. Proficz, "Dynamic gpu power capping with online performance tracing for energy efficient gpu computing using depo tool," *Future Generation Computer Systems*, vol. 145, pp. 396–414, 2023.

[15] D. Szmidka, A. Krzywaniak, P. Czarnul, and J. Proficz, "Dynamic energy-performance optimization tool extension with periodic power cap tuning," in *2025 IEEE 31th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 1–4, 2025.