CEEMS: A Resource Manager Agnostic Energy and Emissions Monitoring Stack

Mahendra Paipuri

Institut du Développement et des Ressources en Informatique Scientifique (IDRIS)
National Centre for Scientific Research (CNRS)
Orsay, France
mahendra.paipuri@cnrs.fr

Abstract—With the rapid acceleration of ML/AI research in the last couple of years, the energy consumption of the Information and Communication Technology (ICT) domain has rapidly increased. As a major part of this energy consumption is due to users' workloads, it is evident that users need to be aware of the energy footprint of their applications. Compute Energy & Emissions Monitoring Stack (CEEMS) has been designed to address this issue. CEEMS can report energy consumption and equivalent emissions of user workloads in real time for HPC and cloud platforms alike. Besides CPU energy usage, it supports reporting energy usage of workloads on NVIDIA and AMD GPU accelerators. CEEMS has been built around the prominent open-source tools in the observability eco-system like Prometheus and Grafana. CEEMS has been designed to be extensible and it allows the Data Center (DC) operators to easily define the energy estimation rules of user workloads based on the underlying hardware. This paper explains the architectural overview of CEEMS, data sources that are used to measure energy usage and estimate equivalent emissions and potential use cases of CEEMS from operator and user perspectives. Finally, the paper will conclude by describing how CEEMS deployment on the Jean-Zay supercomputing platform is capable of monitoring more than 1400 nodes that have a daily job churn rate of around 20k jobs.

Index Terms—Energy measurement, emissions, monitoring, HPC, cloud, GPU.

I. Introduction

With the advent of climate change, there is an immediate need to reduce and optimize the energy usage of ICT technologies. This is only possible when end users and Data Center (DC) operators are made aware of their energy and environmental footprint. Various options are available to report energy usage of compute workloads on HPC and cloud platforms. The most accurate and less intrusive method is to use external watt meters. However, this requires installing additional infrastructure which might not be a scalable option. Moreover, external watt meters measure the consumption of the entire compute node and the energy consumption data of different processes/components are not readily available. Software-based power meters have gained a lot of traction in the last decade and they have different levels of measurement granularity. The downside to software-based power meters is that they are specific to each processor and require detailed knowledge of underlying hardware.

Various monitoring platforms have been introduced in the past like Ganglia [1], XDMoD [2], TACC Stats [3] and PIKA [4]. These solutions lack either better visualization tools, GPU support, energy usage metrics at the compute workload level or customizable energy estimation framework. Prometheus has become a *de-facto* solution for storing time series data in a Time Series Database (TSDB) in the cloud landscape. The raw data from different data sources can be ingested into Prometheus TSDB and using recording rules [5], it is possible to estimate the same derived metric using different rules according to the needs and underlying hardware of the DC. In the current context, this means it is possible to ingest energy usage data from different sources like in-band, out-band measurements, watt meters, *etc.,* and operators can use one or more data sources to estimate the energy usage of each compute workload. Grafana proved to be a versatile solution in the visualization landscape that has strong support for Prometheus. Grafana allows operators to further customize the visualization tools using its plugin eco-system [6] to add custom third-party data sources and/or panels.

In the present work, CEEMS [7] which is based on Prometheus and Grafana will be introduced. CEEMS has been designed to be platform agnostic meaning that it can be deployed on an HPC, Openstack or Kubernetes platform that is running on bare-metal to gather energy and emissions data of individual workloads. The rest of the paper is organized as follows: Section II discusses the overall architecture of CEEMS, Section III briefs configuration aspects of CEEMS using its deployment on Jean-Zay supercomputer as an example and finally, Section IV concludes the paper with few pointers to future work.

II. Overview

A. Philosophy

This section explains how different metrics are gathered from different sources in CEEMS.

a) CPU, memory and IO metrics: Most of the resource managers like SLURM, Libvirt (for Openstack), and Kubelet (for Kubernetes) use Linux Control Groups (cgroups) [8] to manage CPU, memory and IO resources for compute workloads. cgroups have different controllers like CPU, memory, IO, CPUset, PID, *etc.,* which control the usage of that resource. Essentially resource managers create a cgroup for each compute workload (a batch job for the HPC platform, a VM for Openstack, and a pod for Kubernetes) with the

resource constraints imposed based on the workload specification. Consequently, cgroups will keep track of accounting information of each workload to ensure the resource usage of a given workload at a given time does not exceed the allocated resources. This accounting information is stored in the `/sys/fs/cgroup` pseudo-filesystem on each compute node. Thus, by walking through the cgroups file system on each compute node, it is possible to gather the accounting information of each compute unit. Types of metrics available by this method depend on the different cgroup controllers that the resource manager supports. The advantage of this approach is gathering workload metrics is very cheap and non-intrusive. Besides cgroups metrics, node-level metrics like total CPU usage, total memory usage, *etc.,* are also collected from `/sys` and `/proc` pseudo-filesystems.

b) Energy metrics: The most common and widely used metric for measuring energy usage is the Running Average Power Limit (RAPL) [9] counters introduced by Intel in Sandy Bridge architecture. RAPL reports energy consumption on different levels or power domains. Server class hardware comes with intra-node devices which include Baseboard Management Controllers (BMC) that can measure the power consumption of the node. These measurements are exposed to the Operating System (OS) using the IPMI-DCMI [10] interface. The advantage of energy measurements reported by IPMI-DCMI is that they report the total power consumption of the entire node, unlike RAPL counters which include only certain components of the node. On the other hand, the IPMI-DCMI command is not suitable to use at a high frequency (even for every few seconds) whereas RAPL counters are available at microsecond granularity.

Both RAPL counters and IPMI-DCMI readings are collected in CEEMS. This gives the operators the freedom to choose which metrics to present based on their underlying hardware. It is also possible to mix both metrics which will be discussed in Section III. Both RAPL and IPMI-DCMI report energy usage at the node level and it is necessary to distribute the total energy consumption among different compute workloads running on the node at a given time. It is very difficult to have an exact energy usage of each process as modern processors are very complex with deep pipelines. Kepler [11] uses a modeling approach by using current energy usage metrics with pre-trained power models to estimate the energy consumption of individual processes. However, CEEMS uses a simple model by distributing the energy consumption based on the relative CPU time of each workload to the total CPU time of the node, which stays a very good approximation. Scaphandre [12] uses a similar approach with RAPL counters to estimate the energy consumption of each process.

c) Emissions metrics: Normally equivalent emissions are estimated based on the energy consumption and the emission factor which is defined as equivalent grams of CO_2 emitted per Kilo Watt Hour energy consumed. Emission factor depends on the current energy mix, *i.e.,* proportion of different sources of energy in the current electricity generation. Energy mix data is dynamic in time and so as are emission factors. OWID provides static emission factor data [13] for different countries estimated based on historical data. Fortunately, more and more countries are providing real-time emission factors like RTE, the French national electricity distributor does for France [14]. Electricity Maps [15], a Copenhagen-based company provides real-time emission factors for a lot of countries and offers free API access for non-commercial usage. Currently, CEEMS gathers static emission factor data from OWID and real-time data from RTE and Electricity Maps.

d) GPU metrics: Currently, CEEMS supports only NVIDIA and AMD GPU accelerators by leveraging the metrics exposed through NVIDIA DCGM and AMD ROCm interfaces, respectively. For most resource managers including SLURM, the indices of GPU devices that are bound to a workload will not be available post-mortem of the workload. Thus, CEEMS collects and stores the map information of workload ID to GPU indices.

B. Components

CEEMS loosely follow microservices architecture where three different components work in tandem to monitor the compute workloads. These components are described below:

a) CEEMS exporter: This is a Prometheus exporter that runs on each compute node of the cluster. Essentially it is an HTTP server that runs on the node and sends the metrics response to every request in a format understandable by Prometheus. It is very lightweight consuming minimum resources. On average the exporter consumes 15-20 MB of memory and each scrape request takes less than 1 microsecond of CPU time. The exporter contains different collectors which can be enabled or disabled based on needs using CLI options. The exporter supports basic auth and TLS to protect it from DoS/DDoS attacks from malicious users. When using GPU clusters, either DCGM exporter [16] or AMD SMI exporter [17] must be deployed alongside the CEEMS exporter to collect GPU metrics.

b) CEEMS API server: As discussed earlier, CEEMS uses Prometheus to store time series metrics of user workloads. For large clusters, the collected data points can quickly become exorbitant with time. Although Prometheus is a highly performant TSDB, it is not suitable to make queries that span a long duration. An example of such a query can be the total energy usage of a given user or a project on a given cluster for all the workloads during the last year. CEEMS API server has been created to address this issue by storing the aggregate metrics of compute units, users and projects. CEEMS API server fetches information from two different sources: the underlying resource manager to get a list of compute workloads and Prometheus to fetch the metrics of these compute workloads. CEEMS API server uses a relational DB based on SQLite to store the compute workloads and their aggregated metrics. CEEMS API server also serves as an abstraction layer for different resource managers by defining a unified DB schema to store compute units of different resource managers. CEEMS

API server's API documentation [18] provides more details on available API endpoints.

c) CEEMS load balancer: Although Prometheus and Grafana proved to be an impeccable pair in monitoring and observability, they lack one crucial element: Access Control. There is nothing preventing a malicious user that has read access on a data source in Grafana to query for the metrics of *any* compute workload. This is an undesirable consequence from both security and privacy perspectives. CEEMS Load Balancer (LB) has been created to address this issue. It has the dual functionality of being a reverse proxy and load balancer to Prometheus. CEEMS LB intercepts the query request to the backend Prometheus instance, retrieves the workload unique identifier and checks with the CEEMS API server if the workload belongs to the user who initiated the request. The current user is identified using `X-Grafana-User` header that Grafana sends with every query to the backend data source [19]. If the check is successful, CEEMS LB will pass the request to the backend Prometheus and serve the response. As a load balancer, it can perform load balancing among multiple Prometheus instances using classic strategies like round-robin and least connection. Thus, CEEMS LB provides the missing access control element through its reverse proxy role.

C. Architecture

Fig. 1 shows a reference architecture of CEEMS in the context of an HPC platform with SLURM as a batch scheduler. Compute nodes of the cluster are represented on the right-hand side where CEEMS and DCGM/AMD exporters are running. A hot TSDB instance will scrape these compute nodes at a configured interval and ingest the metrics of compute workloads on a local disk. Prometheus does not advise to use network file system as a storage backend due to broken POSIX locks implementations on most of the network file systems. This hot TSDB will replicate the data to Thanos [20], which provides long-term storage capabilities to Prometheus.

As discussed above, the CEEMS API server fetches the job data from SLURM DBD periodically and populates its own DB based on SQLite. At the same time, the CEEMS API server estimates the aggregate metrics by querying Thanos. It is possible to configure the CEEMS API server to clean up TSDB by removing metrics of workloads that did not last more than the configured cutoff time. This helps in reducing the cardinality [21] of metrics. Optionally, SQLite DB can be backed up continuously onto long-term storage using Litestream [22]. CEEMS API server also supports an in-built punctual backup solution at a configured interval.

CEEMS LB sits in front of Thanos to introspect the query before deciding whether to proxy the query or not. It does so by directly querying the CEEMS API server's DB, when available. If the DB file is not accessible, CEEMS LB makes an API request to the CEEMS API server to check the user's ownership status to the queried workload.

Finally, Grafana dashboards can be built using Prometheus and CEEMS API servers as data sources. Prometheus data source provides the time series metrics of each compute workload whereas the CEEMS API server data source serves the aggregated metrics and list of compute workloads.

D. Technical details

The entire CEEMS stack has been implemented in Golang to take advantage of the concurrency model of the programming language. The ease of cross-compiling in Golang is another motivation to choose the language where pre-compiled binaries can be used directly on the target platforms. CEEMS supports different installation methods including containerized deployments, packaged OS and Ansible roles. More details can be found in the installation section of docs [23]. SQLite has been chosen as Relational Data Base Management System (RDBMS) for its simplicity and lack of external dependencies. Moreover, the CEEMS API server do not need concurrent write access as there is only one go routine that writes to DB at a configured interval. This justifies the use of SQLite in the current context. All the CEEMS components can be configured in a single YAML file where each component will read its relevant configuration. All CEEMS components support basic auth and TLS.

III. Deployment on Jean-Zay

CEEMS has been deployed on the Jean-Zay supercomputer, which is a heterogeneous system with approximately 1400 compute nodes (Intel and AMD). There are more than 3500 NVIDIA GPUs (V100, A100 and H100) distributed among different partitions of the system. Most of the compute nodes are diskless and contain a mix of Infiniband and Omni-Path networking.

A. Job energy usage estimation

As discussed in Section I, an important feature of CEEMS is configurability and customization of energy estimation of compute workloads. CEEMS exporter exports the energy consumption metrics reported by both IPMI-DCMI and RAPL on all compute nodes of Jean-Zay. On Intel compute nodes RAPL reports energy usage counters for both CPU and DRAM whereas on AMD compute nodes, only CPU energy counters are reported by RAPL. There are at least two different types of GPU servers. In one server type, IPMI-DCMI reports the total power usage of the compute node including the power usage by GPUs where whereas the other type does not include GPU power usage in the IPMI-DCMI readings. Each of these cases can be handled elegantly using Prometheus by grouping them in different scrape target groups and defining the recording rules accordingly.

For instance, when both RAPL CPU and DRAM energy counters are available, the power usage of each job is estimated using the following approach. As most of the compute nodes of Jean-Zay are diskless, the power consumption of local storage is assumed to be zero. Furthermore, 10% of the total energy consumption is attributed to network [24] and the rest to CPU and DRAM. The energy consumption of other components like memory controllers, registers and PCIe devices are included
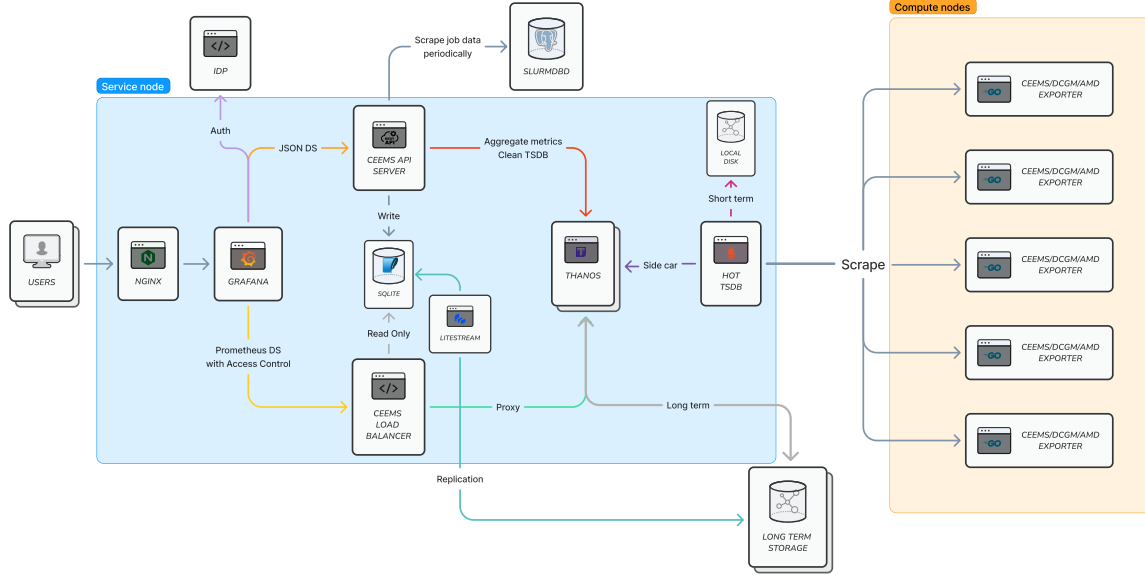
Fig. 1: CEEMS architecture using Thanos and Litestream.

in the consumption of CPU and DRAM. Power usage of a given job is estimated as in (1)

$$
\begin{aligned}
P_{job,t} = {} & 0.9 * P_{ipmi,t} * \left(\frac{P_{rapl,cpu,t}}{P_{rapl,cpu,t} + P_{rapl,dram,t}} \right) * \left(\frac{T_{job,t}}{T_{node,t}} \right) \\
& + 0.9 * P_{ipmi,t} * \left(\frac{P_{rapl,dram,t}}{P_{rapl,cpu,t} + P_{rapl,dram,t}} \right) * \left(\frac{M_{job,t}}{M_{node,t}} \right) \\
& + 0.1 * P_{ipmi,t} * \left(\frac{1}{N_{jobs,t}} \right)
\end{aligned}
\tag{1}
$$

where $P_{job,t}$ is power usage of job, $P_{ipmi,t}$ is power reported by IPMI for entire node, $P_{rapl,cpu,t}$ and $P_{rapl,dram,t}$ power readings from RAPL counters for CPU and DRAM, respectively, $T_{job,t}$ is CPU time of job, $T_{node,t}$ is total CPU time of node, $M_{job,t}$ is memory usage of job, $M_{node,t}$ is memory usage of entire node and $N_{job,t}$ number of jobs running on the node. All the variables are defined at a given time t.

The first two terms in (1) represent the CPU and DRAM power usage of the job, respectively. The total power reported by IPMI is distributed between CPU and DRAM using RAPL counters. Then this total CPU and DRAM power usage is distributed among the running jobs based on the CPU time and memory usage of each job relative to the total CPU time and memory usage of the node. The final term represents the power usage by the network devices. CEEMS exporter does not export any network-related statistics at the moment and hence, the total power usage by networking is distributed equally among the running jobs at a given time. If power usage from other components like distributed storage, network infrastructure, cooling, *etc.,* are available they can be easily integrated into the power estimation. This estimation formula can be customized based on underlying hardware using different Prometheus recording rules for different compute node groups. Example recording rules for different cases are provided in the `etc/prometheus` folder of CEEMS GitHub repository [7].

B. Metrics visualization

CEEMS expose a variety of metrics, both time series and aggregated, which can help end users to keep track of either energy and environmental footprint and optimize the workloads. For example, Fig 2 shows some of the dashboards built based on CEEMS on the Jean-Zay platform. Fig 2a shows the average CPU and GPU and their memory usage, total energy usage and resulting equivalent emissions of a user during the last 3 months. Fig 2b lists SLURM jobs of a user along with aggregated metrics of each job. Finally, Fig 2c shows the time series CPU metrics of a given job.
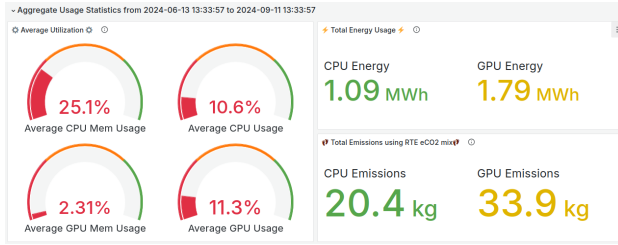
Cluster operators can have similar data available to them, albeit, for the entire cluster. This enables the operators to perform data analysis on the job metrics data to optimize the cluster usage, identify users and/or projects that are using the cluster resources inefficiently, *etc.*

IV. FUTURE WORK

CEEMS is still in the early development phase where initial feedback has been very encouraging. Some of the important features in the pipeline are adding network and IO stats to CEEMS exporter using extended Berkley Packet Filtering (eBPF) framework and adding performance metrics like FLOPS, caching, and memory IO bandwidth to name a few from Linux's `perf` framework. Extending CEEMS to Openstack and Kubernetes is also a long-term objective.
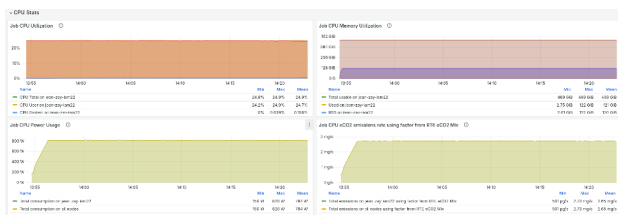
ACKNOWLEDGMENT

(a) Aggregate usage metrics

(b) List of SLURM jobs with aggregate metrics

(c) Time series CPU metrics for a given job

Fig. 2: Grafana dashboards of CEEMS.

in providing insightful and useful feedback on CEEMS which helped to improve the overall CEEMS architecture.

REFERENCES

[1] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817–840, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819104000535

[2] J. T. Palmer, S. M. Gallo, T. R. Furlani, M. D. Jones, R. L. DeLeon, J. P. White, N. Simakov, A. K. Patra, J. Sperhac, T. Yearke, R. Rathsam, M. Innus, C. D. Cornelius, J. C. Browne, W. L. Barth, and R. T. Evans, "Open XDMoD: A tool for the comprehensive management of high-performance computing resources," *Computing in Science & Engineering*, vol. 17, no. 4, pp. 52–62, 2015.

[3] T. Evans, W. L. Barth, J. C. Browne, R. L. DeLeon, T. R. Furlani, S. M. Gallo, M. D. Jones, and A. K. Patra, "Comprehensive resource use monitoring for HPC systems with TACC stats," in *2014 First International Workshop on HPC User Support Tools*, 2014, pp. 13–21.

[4] R. Dietrich, F. Winkler, A. Knüpfer, and W. Nagel, "PIKA: Center-wide and job-aware cluster monitoring," in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, 2020, pp. 424–432.

[5] Prometheus. (2024) Defininig recording rules. [Online]. Available: https://prometheus.io/docs/prometheus/latest/configuration/recording_rules/

[6] Grafana. (2024) Grafana plugin tools. [Online]. Available: https://grafana.com/developers/plugin-tools/

[7] M. Paipuri, "Compute Energy & Emissions Monitoring Stack (CEEMS)," https://github.com/mahendrapaipuri/ceems, 2024.

[8] *Control Groups*, Linux Kernel Community, 2024, version 6.11.0. [Online]. Available: https://docs.kernel.org/admin-guide/cgroup-v1/cgroups.html

[9] *Intel 64 and IA-32 Architectures Software Developer Manuals*, Intel, 2024, volume 4.

[10] Intel. (2024) DCMI data center manageability interface specification. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/dcmi-v1-5-rev-spec.pdf

[11] Kepler. (2024) Kepler power model. [Online]. Available: https://sustainable-computing.io/design/power_model/

[12] Hubblo, "Scaphandre," https://github.com/hubblo-org/scaphandre, 2024.

[13] OWID. (2024) CO_2 and greenhouse gas emissions data explorer. [Online]. Available: https://ourworldindata.org/co2-and-greenhouse-gas-emissions#explore-data-on-co2-and-greenhouse-gas-emissions

[14] RTE. (2024) eco2mix - CO_2 emissions per kwh of electricity generated in france. [Online]. Available: https://www.rte-france.com/en/eco2mix/co2-emissions

[15] ElectricityMaps. (2024) API documentation. [Online]. Available: https://static.electricitymaps.com/api/docs/index.html

[16] NVIDIA, "DCGM exporter," https://github.com/NVIDIA/dcgm-exporter, 2024.

[17] AMD, "AMD SMI exporter," https://github.com/amd/amd_smi_exporter, 2024.

[18] M. Paipuri. (2024) CEEMS API documentation. [Online]. Available: https://mahendrapaipuri.github.io/ceems/api

[19] GrafanaLabs. (2024) Configure grafana. [Online]. Available: https://grafana.com/docs/grafana/latest/setup-grafana/configure-grafana/#send_user_header

[20] Thanos. (2024) Thanos. [Online]. Available: https://thanos.io/

[21] V. T. Archana Kesavan. (2022) How to manage high cardinality metrics in prometheus and kubernetes. [Online]. Available: https://grafana.com/blog/2022/10/20/how-to-manage-high-cardinality-metrics-in-prometheus-and-kubernetes/

[22] Litestream. (2024) Litestream. [Online]. Available: https://litestream.io/

[23] M. Paipuri. (2024) CEEMS documentation. [Online]. Available: https://mahendrapaipuri.github.io/ceems/docs/

[24] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 732–794, 2016.