Studying the Energy Consumption of Stream Processing Engines in the Cloud

Govind KP, Guillaume Pierre, Romain Rouvoy

Studying the Energy Consumption of Stream Processing Engines in the Cloud

Govind KP
Univ. Rennes, Inria, CNRS, IRISA
Rennes, France

Guillaume Pierre
Univ. Rennes, Inria, CNRS, IRISA
Rennes, France

Romain Rouvoy
Univ. Lille, Inria, CNRS, UMR 9189 CRIStAL, France

***Abstract*—Reducing the energy consumption of the global IT industry requires one to understand and optimize the large software infrastructures the modern data economy relies on. Among them are the data stream processing systems that are deployed in cloud data centers by companies, such as Twitter, to process billion of events per day in real time. However, studying the energy consumption of such infrastructures is difficult because they rely on a complex virtualized software ecosystem where attributing energy consumption to individual software components is a challenge, and because the space of possible configurations is large. We present GreenFlow, a principled methodology and tool designed to automate the deployment of energy measurement experiments for data stream processing systems in cloud environments. GreenFlow is designed to deliver reproducible results while remaining flexible enough to support a wide range of experiments. We illustrate its usage and show in particular that consolidating a DSP system in the smallest number of servers that are capable of processing it is an effective way to reduce energy consumption.**

***Index Terms*—Green computing, data stream processing, energy consumption, reproducibility.**

I. Introduction

The energy used to power the global data center industry is currently 200 TWh every year, and this number is expected to increase up to 280 TWh in the next 4-5 years [1]. On the other hand, the latest objectives from the United Nations' Climate Change Conference are to *reduce* the global greenhouse gas emissions by 40% relative to the 2019 level [2]. To help reach this ambitious objective, numerous research efforts are underway to understand and curb the energy consumption of our data centers [3].

Reducing the ecological impact of the IT industry without compromising the provided quality of service is very difficult. At the hardware level, data center operators aim to improve their energy efficiency and reduce their costs by minimizing their *Power Usage Effectiveness* (PUE). However, these efforts are now reaching their point of diminishing returns [4]. Modern data centers have a PUE around 1.1, very close to the theoretical minimum of 1 [5]. Reducing energy consumption further requires one to study and optimize the software which runs in these servers — from the low-level operating system to the cloud management layer, the middleware frameworks and eventually to the end-user applications themselves. Energy consumption is all-encompassing, and any efficiency improvement in one layer of the architecture can be nullified by inefficiencies in the other layers [6].

One of the major drivers of increasing energy demands for cloud is the enormous and continuously increasing amount of data that is being generated and processed. The increasing demands of our data economy have spawned a rich landscape of technologies, approaches, and paradigms to manage the volume, variety, and velocity of big data [7]. In particular, *Data Stream Processing* (DSP) systems have become increasingly popular as they provide significant advantages in dealing with very large data sizes (Volume) while enabling real-time processing of data sets that are being dynamically generated continuously (Velocity) [8].

DSP systems operate on a continuous stream of data that are processed incrementally as they arrive. The data are usually processed in small, fixed-size time intervals called windows, which enables real-time analysis and decision-making. This approach differs from the traditional paradigm of batch processing, where data are processed in large, discrete batches, which can result in delays and missed opportunities for real-time insights and actions.

Stream processing is a highly adaptable and resilient paradigm, which makes it an attractive option for various use cases across different fields. Stream processing engines such as Flink and Heron, along with the Kafka message bus, form the backbone of data processing workflows at major companies, capable of processing billions of events per day in real time [9]. The sheer scale and widespread adoption of DSP systems also makes them a high-impact target for energy optimization — even incremental improvements could translate to significant amounts of energy and cost savings.

However, there is still a lot of groundwork that needs to be laid in order to understand and optimize the energy costs of these systems. To move towards energy efficiency in these systems, we need to first move towards reliable and reproducible (or at least interpretable [10]) energy measurements of these systems. This is very challenging due to the following reasons:

- Production deployments of DSP systems are distributed across multiple servers to ensure availability and reliability, generally managed with a resource manager like YARN or Kubernetes, on multi-tenant and possibly multi-site environments. This means that node-level metrics are usually not detailed enough to understand the energy consumption of these systems. Obtaining finer-grained attribution of the energy spent to individual software components remains a difficult challenge.

- DSP systems are complex systems with several interdependent components. For instance, running the Apache Flink DSP engine requires the coordination of multiple TaskManagers and JobManagers running across multiple nodes. Data ingestion is usually handled using a distributed Apache Kafka cluster. This is in turn coordinated with the help of a Zookeeper cluster running across multiple nodes for fault-tolerance reasons. The whole setup also needs access to a distributed storage system and is typically deployed using a container orchestrator such as Kubernetes. Setting up an experimental benchmark in order to investigate the energy costs of DSP systems requires a significant amount of orchestration and setup.
- DSP systems are quite complex with several subsystems and hundreds of tunable parameters for each of these subsystems [11]. These parameters can have a significant effect on energy consumption. This parameter space is quite large and challenging to tackle.
- DSP systems are highly dynamic and adaptive, capable of scaling elastically depending on the workload. This results in highly variable resource usage patterns, further complicating efforts to accurately measure energy consumption.

To tackle these challenges, any enquiry into the energy impact of DSP systems must first solve the problem of distributed energy measurement with fine-grained attribution. The large parameter space and the dynamic nature also necessitates a robust experimental testbed to be able to run reproducible benchmarks.

There have been previous attempts to investigate energy-efficiency in DSP systems, proposing innovative approaches in scheduling [12] for specific implementations, as well as comparing the performance efficiency of different implementations [13]. However, even under ideal situations, running small-scale experiments for the purpose of benchmarking, energy measurement is fraught with many possible pitfalls [14], [15]. A key challenge in studying the energy characteristics of stream processing systems is the lack of standardized benchmarks and metrics for energy efficiency. Previous solutions have tended to prioritize optimization over reproducibility and documentation, under-specifying the details of their experimental setup. This makes it difficult to compare the energy performance of different systems and to identify areas for improvement.

This paper proposes a robust methodology based on the *Design Of Experiments* model [16], as well as GreenFlow, an experimental testbed to tackle the unique challenges of energy-measurement of DSP systems. GreenFlow provides an end-to-end solution for deploying and managing the experiment lifecycle, as well as providing tools for dealing with the exponential complexity in the parameter space. It exploits the best practices in energy measurement and makes use of the Scaphandre [17] software power meter, which can attribute energy at fine granularity with high level of accuracy [18]. The testbed is built on the Grid'5000 bare-metal platform, which permits a high amount of reproducibility [19]. This approach is generalizable to study the energy consumption of any cloud-native distributed system.

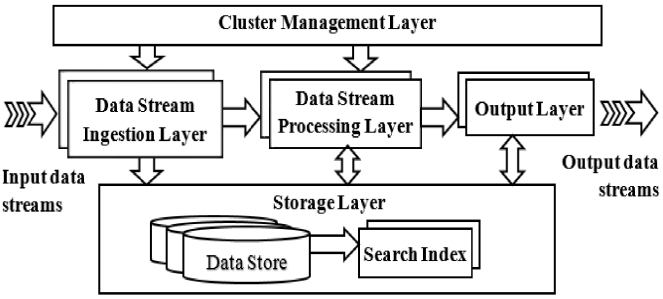

Fig. 1. Architecture of a typical data stream processing system (adapted from [21]).

We demonstrate the usage and flexibility of GreenFlow by evaluating the performance tradeoffs of specific workloads from the Theodolite benchmark suite [20]. We show in particular that consolidating a DSP system in the smallest number of servers that are capable of processing it is an effective way to reduce energy consumption.

The remainder of the paper is organized as follows: Sections II and III discuss the technological background and related works, respectively. Section IV presents GreenFlow's methodology and tool. Section V evaluates our framework and, finally, Section VI concludes this paper.

II. Background

A. DSP systems

DSP (*Data Stream Processing*) systems are designed to process continuous streams of data in near real-time. The fundamental operators in stream processing systems include filtering, mapping, aggregating, joining, and windowing, while the operands are the data elements within the streams. These operators are applied to the data elements in a streaming manner, allowing for low-latency processing of large volumes of data.

These systems are composed of multiple key components [21]. As shown in Figure 1, the ingestion layer is in charge of collecting data from various sources. The processing layer is where actual data processing occurs. The output layer delivers the processed data to the outside world, whereas the storage layer stores intermediate versions for further analysis or retrieval. The cluster management layer oversees hardware and infrastructure resources.

Data stream ingestion is usually handled by a separate system, such as Apache Kafka [22], whereas popular data stream processing engines include Apache Flink [23], Kafka Streams [24], and Apache Spark [25]. Each engine has its strengths and weaknesses, such as differing levels of parallelism, fault tolerance, and API support, which must be considered when selecting the appropriate engine for a specific use case. One may also expect that different engines feature different energy consumption profiles. In real-world

production deployments, stream processing systems are typically deployed using *virtual machines* (VMs) or container orchestration platforms, like Kubernetes, for scalability and resource management [26].

The parameter space of DSP systems is wide, due to the numerous configuration options available for the different layers. these options include parameters like parallelism, buffer sizes, window sizes, and backpressure settings. Furthermore, managing the parameter space effectively is difficult, as it is highly workload-sensitive and requires deep understanding of the trade-offs between performance and resource usage [27]. It is, therefore, the purpose of GreenFlow to enable and facilitate the exploration of this parameter space.

B. Energy Metrology

The energy consumption of any software system may vary greatly depending on contextual factors such as temperature, workload, and the type of hardware used, thus highlighting the importance of accurately measuring and understanding energy usage to optimize the energy efficiency.

The most accurate measurement method of energy consumption is the wattmeter reading, which is the physical measurement of the instantaneous power at a server's socket. However, measuring socket power usage requires specialized hardware devices, such as power meters and smart plugs, and it does not capture the fine-grained energy usage of individual software components within the system. An alternative is to use software power meters, which can provide a more fine-grained view, yet possibly with lower levels of accuracy [28].

Software power meters exploit special hardware performance counters provided by modern CPUs, which give fine-grained indications about CPU and memory resource usage. These metrics are made available using the RAPL (Running Average Power Limit) library [29]. Software power meters then utilize a variety of models to derive an estimation of the marginal energy usage that can be attributed to each software process in the machine [30].

Among the several available software power meters, we decided to use Scaphandre in this work [17]. Besides the ability to estimate the energy consumption of the entire hardware machine as well as individual processes, Scaphandre can also measure the energy consumption of virtual machines and containers that it can then attribute to high-level concepts such as Kubernetes pods. It then exposes these power consumption metrics as a Prometheus exporter, enabling meaningful measurements across an entire distributed Kubernetes cluster.

III. Related Works

There have been numerous studies examining the runtime characteristics of DSP systems. These studies can be broadly classified into "horizontal" and "vertical" studies. Horizontal studies involve comparing different implementations, while vertical studies delve deeper into a specific optimization approach for a particular implementation. Both types of studies are crucial for understanding the performance and energy efficiency of DSP systems.

The complexity of stream processing systems makes horizontal studies difficult to execute due to the intricacies of the experimental setup, lifecycle, and various components that require alignment for a single experiment. Furthermore, such studies necessitate standardised benchmarks and workloads to enable meaningful comparisons between different DSP systems. Unfortunately, the absence of these standards in this domain complicates reproducibility and comparison.

[13] is, to the best of our knowledge, the only horizontal study. The paper describes a benchmarking study comparing the energy consumption of different DSP engines (Storm, Esper, S4, and Spark). They use the *Power Distribution Unit* (PDU) for power measurement and report interesting findings: energy efficiency in such a system may not necessarily be correlated with the CPU usage.

On the other hand, vertical studies have been more frequent, investigating specific optimization approaches for particular DSP engines [31], [32]. For example, [12] proposes a dynamic energy-efficient scheduling approach for Apache Storm, while [33] proposes an hybrid execution model for energy-efficient stream processing in Apache Spark. However, the studies above did not focus on cloud-native deployments, an increasingly dominant trend in DSP systems.

Recently, an innovative cloud-native Kubernetes implementation, Theodolite, was proposed for benchmarking stream processing systems [20]. This framework is unique in its extensibility and can be leveraged to support both horizontal and vertical studies. This is crucial in providing recommendations regarding trade-offs between DSP engines as well as workload-specific tuning that can now be studied in detail.

Measuring energy in the general case is fraught with many possible pitfalls [15]. Current research lacks robust, repeatable methodologies for energy consumption studies in real-world, cloud-native deployment settings, leading to a gap in the field. There has been a massive shift towards more cloud-native deployments for production DSP systems, generally supported by a container orchestration layer like Kubernetes. This means that, in practice, we should look at the overhead of the container runtime, as well as the inseparable infrastructure costs of the control plane.

In this work, we propose a comprehensive solution for measuring the energy consumption of DSP systems in a modern cloud-native environment, with Kubernetes as the orchestration layer. Our work addresses the unmet need for energy efficiency insights across different DSP engines under production-like environments at scale. We hope that this helps us develop a more nuanced understanding of energy consumption in complex cloud systems.

IV. Methodology

When conducting any sort of evaluation or benchmarks of complex systems, it is essential to clearly define the purpose of the evaluation and identify the fundamental requirements of the system being evaluated. We therefore aim at supporting rigorous experimental procedures and data analysis following

1) **Requirement Recognition:** Recognize the problem, and state the purpose of a proposed evaluation.
2) **Service Feature Identification:** Identify Cloud services and their features to be evaluated.
3) **Metrics and Benchmarks Listing:** List all the metrics and benchmarks that may be used for the proposed evaluation.
4) **Metrics and Benchmarks Selection:** Select suitable metrics and benchmarks for the proposed evaluation.
5) **Experimental Factors Listing:** List all the factors that may be involved in the evaluation experiments.
6) **Experimental Factors Selection:** Select limited factors to study, and also choose levels/ranges of these factors.
7) **Experimental Design:** Design experiments based on the above work. Pilot experiments may also be done in advance to facilitate the experimental design.
8) **Experimental Implementation:** Prepare experimental environment and perform the designed experiments.
9) **Experimental Analysis:** Statistically analyze and interpret the experimental results.
10) **Conclusion and Reporting:** Draw conclusions and report the overall evaluation procudure and results.

Fig. 2. Cloud services evaluation methodology (adapted from [16]).

the CEEM methodological principles [16] (also highlighted in Figure 2).

In the specific case of evaluating energy measurement in stream processing systems, we aim to answer the following research questions:

1) Given a specific stream processing workload, how does the energy usage distributes across software components?
2) How can experimental procedures be designed to ensure rigorous and comprehensive evaluation of energy consumption across different DSP systems?

Answering these questions requires one to design an experimental framework that is both highly configurable (to support a wide range of experiments) and rigorous (to provide meaningful and reproducible results). We first discuss the principles behind GreenFlow's design, then detail the different components of the system architecture.

A. Design principles

GreenFlow is designed to exploit the capabilities of an advanced bare-metal cloud architecture such as Grid'5000 [19] which allows its users to provision specific hardware resources using short-duration reservations. Any single experiment therefore initially provisions one or more servers and installs a virtual resource orchestrator. It then deploys the required software stack including an DSP engine with its data storage and configuration management services (typically Apache Flink [23] and Zookeeper [34]), a data ingestion and output layer (typically Apache Kafka [22]), a software power meter (Scaphandre [17]) and a benchmark application with its associated load injector (Theodolite [20]). It configures an external monitoring system (Prometheus [35]) to collect the relevant metrics from the respective deployed system components, and triggers the load injector to actually start the experiment. Once the experiment has completed it then tears down the deployed software and releases the provisioned servers.

1) Declarative configurable deployment: With a large number of software components that each can be deployed and configured in a large number of ways, GreenFlow needs to be highly configurable in order to reliably instantiate a large number of possible configurations. Any critical function is helping the user to navigate this complexity in the parameter space. Instead of designing GreenFlow as an imperative program such as a Python script, we rather use a declarative approach based on Ansible [36] which allows us to describe the system state that we want to obtain rather than the means to reach this state.

2) Cloud native design: Executing any experiment requires one to deploy a significant number of inter-related software components. To accurately reproduce the execution environment and performance characteristics of a DSP system in a real cloud data center, these tools should exploit modern cloud infrastructure and leverage cloud-specific capabilities such as auto-scaling, load balancing, and high availability. We therefore designed GreenFlow to follow a cloud-native approach based on the popular Kubernetes container orchestrator [37], enabling accurate measurement of energy consumption in a production-like environment.

3) Best practices for reproducibility: An important goal of our work is to make experiments as reproducible as possible. In particular, energy measurements are known to be difficult to reproduce, with potential large variance in energy measurements between "identical" executions [38]. We therefore leverage best practices and processes from modern software engineering such as pinning all the dependencies of the system, and running the entire code from a Docker container [39].

B. System Design

As presented in Figure 3, GreenFlow deploys the full necessary software stack for experiments in hardware machines provisioned using a bare-metal cloud platform such as Grid'5000 [19]. It stores the measured metrics outside of this platform so the collected data remains available after releasing the bare-metal resources. We now discuss the system design relating to configurability and reproducibility, fine-grained energy measurements, metrics management, and workload injection.

1) Configuration management: An important goal of GreenFlow is to allow users to explore a large number of configuration options. The configuration space is very large due to the large number of tunable parameters in the system, and many of these parameters are likely to have an impact on the DSP system performance and/or energy consumption [27]. This necessitates providing a seamless way to inject parameters into the system, as well as capture the list of parameters for each experiment run for retrospection and further analysis.

GreenFlow makes use of the Gin Config framework, originally developed by Google engineers for managing the hyperparameter space in machine learning models [40]. This enables

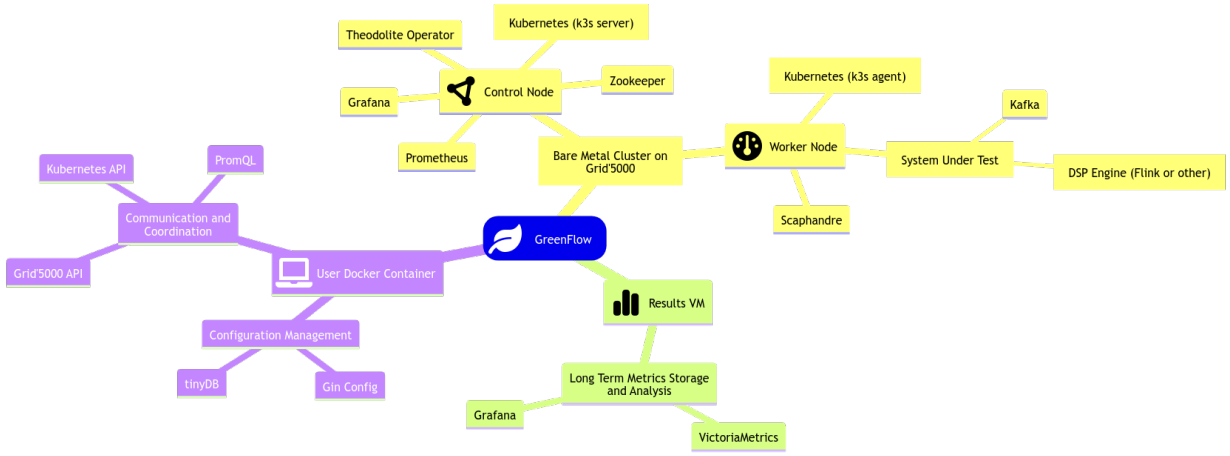

Fig. 3. GreenFlow's architecture.

```
greenflow.g5k.G5KPlatform.site = "lyon"
greenflow.g5k.G5KPlatform.cluster = "nova"
greenflow.g5k.G5KPlatform.num_control = 1
greenflow.g5k.G5KPlatform.num_worker = 1
greenflow.g5k.G5KPlatform.queue = "default"
greenflow.g5k.G5KPlatform.walltime = "1:00:00"

greenflow.factors.kafkaOnWorker = True
greenflow.factors.uc1_flink_resourceValue = 1
greenflow.factors.uc1_flink_loadValue = 10000
greenflow.factors.uc3_flink_resourceValue = 1
greenflow.factors.uc3_flink_loadValue = 10000
greenflow.factors.duration = 300
```

Fig. 4. Example GreenFlow configuration file.

a dependency injection framework for dynamically injecting parameters at runtime. Additionally, GreenFlow uses tinyDB as a lightweight single-file database for experiment metadata storage, allowing for easy association of specific experiment runs with their corresponding parameters [41].

Any factors that can affect the experiment are specified in a config file. There can be multiple profiles of config files that enables mix-and-match at runtime. At the same time, many parameters can use default values, which reduces typical configuration files to a handful of simple parameters. An example configuration file specifying the site, cluster, number of control and worker nodes, and the duration of an experiment is shown in Figure 4.

2) Energy measurement: In any GreenFlow experiment, a large number of software components get deployed in the provisioned servers. This necessitates being able to attribute the energy consumed by these servers to individual processes. Because the experiments get deployed using Kubernetes, it is useful to be able to trace the estimated per-process consumed energy to individual Kubernetes pods, which in turn can be easily attributed to different software components.

For this reason we make use of the Scaphandre software power meter [17]. Scaphandre exploits hardware-level resource utilization counters translated to energy consumption estimations by the Intel RAPL toolkit. It then provides comprehensive, pod-level metrics. Scaphandre can be deployed on every node from the experiment as a Kubernetes DaemonSet to collect metrics on the whole cluster.

To measure energy consumption during the experiments, we make use of Prometheus to scrape a Scaphandre exporter with a sampling period of 5 seconds, which provides near real-time energy consumption metrics at the process level. It is also possible to match this to Kubernetes metadata such as the pod and namespace. Since Grid'5000 machines are ephemeral by nature, these metrics are then exported in a long-term storage virtual machine running VictoriaMetrics [42].

3) Metrics management: Aside from the energy measurements coming from Scaphandre, there are many other useful metrics to understand the performance of the DSP system and identifying potential bottlenecks. These include the actual DSP engine metrics produced by Flink and Kafka. As the core node-level metrics like CPU usage, I/O, network throughput may also prove to be very insightful for understanding these systems, we deploy the Prometheus node-exporter on each node to collect these metrics.

These metrics can later be retrieved later by leveraging the PromQL (Prometheus Query Language) API, enabling further analysis [43]. For visualization and exploration of the data, a Grafana dashboard is also provisioned [44].

4) Experiment lifecycle: Resources are provisioned from the Grid'5000 API through the EnOSLib library [45]. This definition defines an OAR job that reserves some bare-metal nodes for the lifetime of the experiment. The node description (along with the entire software stack) at the time of the experiment is collected and stored along with results for interpretability.

A lightweight Kubernetes distribution is deployed on these nodes. The Kubernetes cluster thus created has a single controller node. To minimize the effects of the noisy neighbor pods from affecting the results of the experiment, no stream processing-related workloads are run on the controller node, and it is reserved for running the supporting infrastructure for

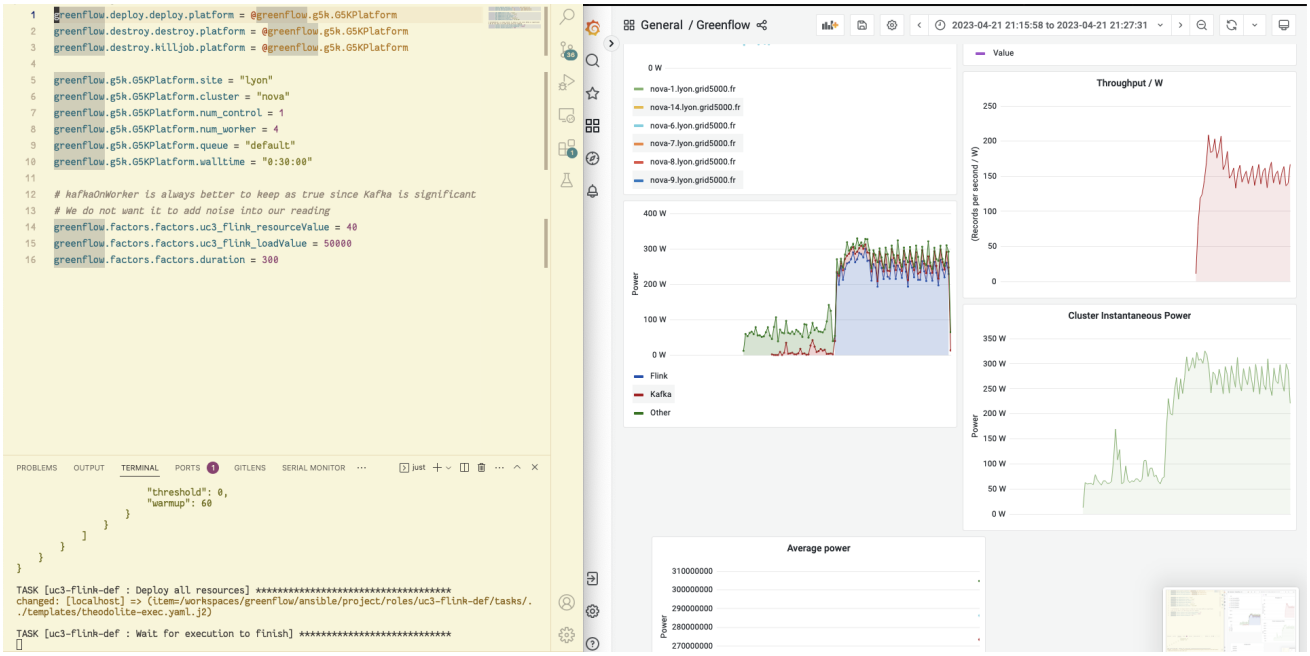

Fig. 5. Greenflow's dashboard example.

the experiment. The worker nodes, on the other hand, only run the lightweight k3s-agent and are tasked with running the core stream processing workloads.

After the Kubernetes cluster is fully initialized, the manifests for the supporting infrastructure for the experiment are deployed. This includes the Prometheus monitoring stack, as well as the aforementioned Scaphandre monitoring tool, which comes integrated with an exporter for Prometheus. The Prometheus node-exporter is deployed on all nodes as well to collect node-related system metrics like CPU utilization, memory usage, and disk I/O.

The heart of this platform is the Theodolite framework [20] which provides a number of data stream processing applications with a standardized workload generator. Once Theodolite is started, it takes care of deploying all the DSP-related components such as Kafka, Zookeeper, and Flink.

5) Interface: Figure 5 shows a screenshot of GreenFlow in operation. It is composed of three windows which respectively allow the user to view and edit key parts of the system configuration (here: the Gin configuration file that configures the parameters for the experiment); launch and monitor an experiment; and finally to view key performance and energy-related metrics in quasi real-time while the experiment is running. The experiment deployment scripts and their dependencies are containerized and can therefore easily be launched with a simple command from the user's personal computer. A separate interface allows users to explore historical metrics data from previous executions.

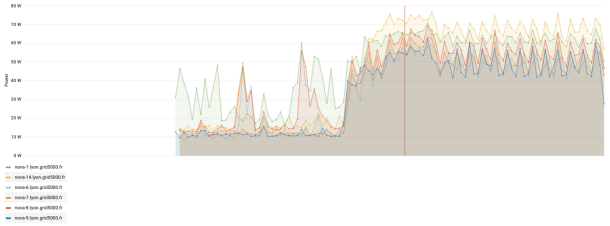

Fig. 6. Node-wise power consumption.

V. Evaluation

A. Experimental Setup

We now showcase GreenFlow's flexibility and effectiveness in providing accurate and detailed energy consumption measurements using a series of experiments in the Grid'5000 testbed. Each server in the "Nova" cluster is equipped with two 8-cores Intel Xeon E5-2620 v4 CPUs, 64 GiB of RAM and a 10 Gb/s Ethernet network interface.

We exercise the system using the Flink implementation of the UC-3 workload from the Theodolite scalability benchmarking suite [20]. This benchmark processes a stream of artificially-generated sensor measurements by splitting the stream according to the sensor ID and implementing time attribute-based aggregation on each stream of individual sensor inputs, a typical type of workload found in IoT systems. In the following studies, we produce a constant flow of 50,000 records/second, and vary the number of server nodes to process this workload.

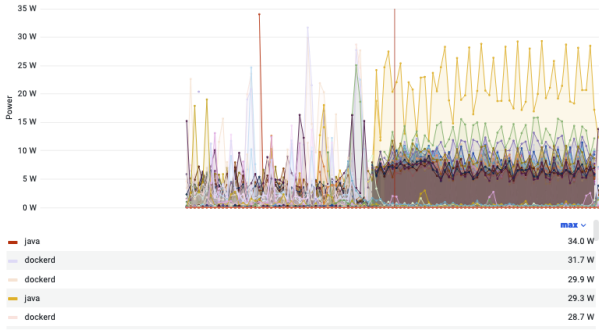

Fig. 7. Process-wise breakdown of power consumption.

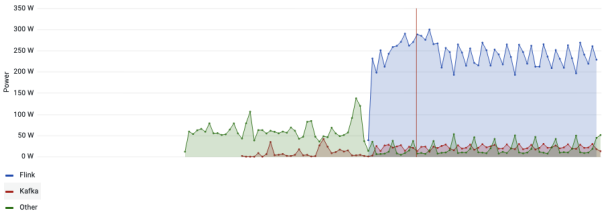

Fig. 8. Component-wise breakdown.

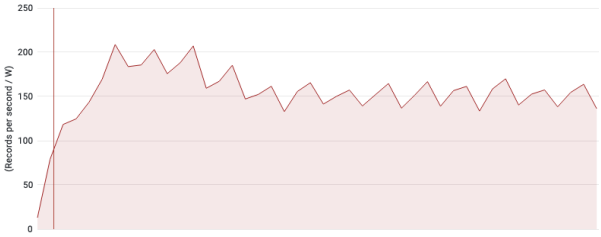

Fig. 9. Throughput per watt for 5 worker nodes.

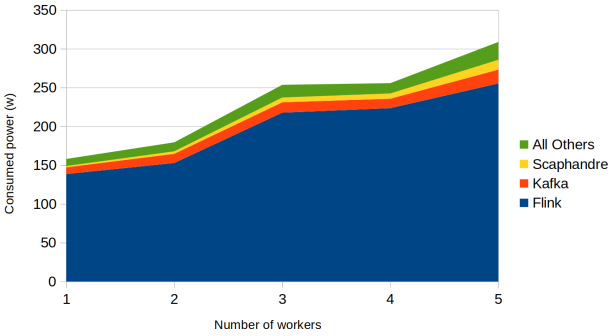

Fig. 10. Impact of consolidation strategies when processing a workload of 50,000 records/second.

B. Detailed energy measurements

GreenFlow allows users to visualize fine-grained energy consumption metrics in real time while an experiment is running, or after the end of the experiment. For example, Figure 6 illustrates the per-node power consumption profile during an execution with 6 server nodes (one control plane and 5 workers). Before the experiment begins, idle servers' energy consumption is low. We can then in particular see the impact of starting the workload on the nodes' energy profile and notice that the energy consumption seems slightly unbalanced among the nodes.

As previously discussed, per-server energy measurements are often too coarse-grained to produce valuable insight into the details of different software components being deployed in each node. Figure 7 shows the energy profiles split on a per-process basis thanks to a simple PromQL query: `scaph_process_power_consumption_microwatts / 10^6`. This query returns more than 4000 individual time series of data, corresponding to the instantaneous power consumed by every process across the entire 6-node cluster.

To derive more meaningful insight, one may aggregate the individual process energy profiles according to the software components they belong to. For example, Figure 8 shows the energy profiles of Flink, Kafka, and other components aggregated across the six nodes of the experimental run.

GreenFlow collects numerous metrics coming from a variety of sources: energy measurements coming from Scaphandre, system-level metrics such as CPU and memory usage coming from "node exporter", data stream processing-level metrics coming from Flink, etc. This provides opportunities to analyze the system's energy efficiency and performance by combining metrics coming from different sources, either in real time or on historical data using the PromQL interface. For example, Figure 9 shows the processed data throughput per watt by dividing the throughput (records per second produced by Flink) by the power consumption (watts per process produced by Scaphandre).

C. Impact of consolidation strategies

To illustrate a typical usage of greenFlow, we now conduct a simple study to evaluate the impact of consolidation strategy on the energy efficiency of the system. In other terms, to process a given workload, is it more interesting to exploit a small number of highly-loaded servers, or a larger number of lightly-loaded ones? Figure 10 shows the breakdown of energy consumption per software component when varying the number of worker nodes from 1 to 5, while keeping the workload constant at 50,000 records/second. We clearly see that Flink's energy usage is one order larger than that of the other components, followed by Kafka, and Scaphandre. We also see that the total energy usage increases when increasing the number of servers, even though each server becomes less and less loaded as a result. This provides clear indication that consolidating the workload in the smallest possible number of servers is an effective way to reduce the system's energy usage. Note that, in the case of a time-varying data streaming workload, keeping the number of servers to a minimum may require the usage of autoscaling techniques [46].

This simple study demonstrates GreenFlow's effectiveness in analyzing the energy usage of stream processing systems, with further in-depth analysis planned for future work. It also demonstrates GreenFlow's capability to provide valuable insights into the energy efficiency of stream processing systems,

which can help system designers and operators optimize the energy usage of their systems.

VI. Conclusion

We have presented GreenFlow, a methodology and tool for studying the energy consumption of data stream processing systems in cloud data center environments. GreenFlow allows a wide range of studies thanks to its configurability while guaranteeing reproducible experiments thanks to robust deployment methods of the complex software stack. Although the many components of a DSP system typically co-exist on a limited number of nodes, GreenFlow can accurately attribute the energy consumption to individual software components. We demonstrated the usage of GreenFlow to conduct a study which demonstrated that the importance of consolidation to save energy in DSP systems. GreenFlow is available in open source[1] in the hope it will enable a wide range of future research studies.

Acknowledgments

This work is supported by the "FrugalCloud" Inria and OVHcloud partnership. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities, as well as other organizations (see https://www.grid5000.fr).

References

[1] G. Kamiya, "Data centres and data transmission networks – analysis - IEA," https://www.iea.org/reports/data-centres-and-data-transmission-networks, Sep. 2022.

[2] European Commission, "EU agrees to COP27 compromise to keep Paris Agreement alive and protect those most vulnerable to climate change*," Press release, Nov. 2022, https://ec.europa.eu/commission/presscorner/detail/%20en/ip_22_7064.

[3] A.-C. Orgerie, M. D. de Assuncao, and L. Lefevre, "A survey on techniques for improving the energy efficiency of large-scale distributed systems," *ACM Computing Surveys*, vol. 46, no. 4, Apr. 2014.

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

[5] "Efficiency – Data Centers – Google," https://www.google.com/about/datacenters/efficiency/.

[6] V. Anand, Z. Xie, M. Stolet, R. De Viti, T. Davidson, R. Karimipour, S. Alzayat, and J. Mace, "The odd one out: Energy is not like other metrics," in *Proc. HotCarbon*, 2022.

[7] S. Uttamchandani, "2022 technology landscape for self-service data," Medium, Jan. 2022, https://modern-cdo.medium.com/2022-technology-landscape-to-democratize-data-272b8228b6cb.

[8] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle, "The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing," *Proceedings of the VLDB Endowment*, vol. 8, 2015.

[9] L. Zhang and D. Malife, "Processing billions of events in real time at Twitter," https://blog.twitter.com/engineering/en_us/topics/infrastructure/2021/processing-billions-of-events-in-real-time-at-twitter-.

[10] T. Hoefler and R. Belli, "Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results," in *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2015.

[11] H. Herodotou, L. Odysseos, Y. Chen, and J. Lu, "Automatic performance tuning for distributed data stream processing systems," in *Proc. IEEE ICDE*, May 2022.

[12] H. Li, H. Dai, Z. Liu, H. Fu, and Y. Zou, "Dynamic energy-efficient scheduling for streaming applications in Storm," *Computing*, vol. 104, no. 2, Feb. 2022.

[13] M. Dayarathna, Y. Li, Y. Wen, and R. Fan, "Energy consumption analysis of data stream processing: a benchmarking approach," *Software: Practice and Experience*, vol. 47, no. 10, 2017.

[14] D. Hackenberg, T. Ilsche, R. Schone, D. Molka, M. Schmidt, and W. E. Nagel, "Power measurement techniques on standard compute nodes: A quantitative comparison," in *Proc. IEEE ISPASS*, Apr. 2013.

[15] Z. Ournani, M. C. Belgaid, R. Rouvoy, P. Rust, J. Penhoat, and L. Seinturier, "Taming energy consumption variations in systems benchmarking," in *Proc. ACM/SPEC ICPE*, Apr. 2020.

[16] Z. Li, L. OBrien, and H. Zhang, "CEEM: A practical methodology for cloud services evaluation," in *Proc. IEEE World Congress on Services*, Jun. 2013.

[17] Scaphandre, 2023, https://github.com/hubblo-org/scaphandre.

[18] M. Jay, V. Ostapenco, L. Lefèvre, D. Trystram, A.-C. Orgerie, and B. Fichel, "An experimental comparison of software-based power meters: focus on CPU and GPU," in *Proc. ACM/IEEE CCGrid*, May 2023, https://hal.inria.fr/hal-04030223.

[19] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, ser. Communications in Computer and Information Science, I. I. Ivanov, M. van Sinderen, F. Leymann, and T. Shan, Eds., 2013, vol. 367.

[20] S. Henning and W. Hasselbring, "Theodolite: Scalability benchmarking of distributed stream processing engines in microservice architectures," *Big Data Research*, vol. 25, Jul. 2021.

[21] H. Isah, T. Abughofa, S. Mahfuz, D. Ajerla, F. Zulkernine, and S. Khan, "A survey of distributed data stream processing frameworks," *IEEE Access*, vol. 7, 2019.

[22] The Apache Software Foundation, "Apache Kafka," https://kafka.apache.org/.

[23] ——, "Apache Flink® – stateful computations over data streams," https://flink.apache.org/.

[24] ——, "Kafka Streams," https://kafka.apache.org/documentation/streams/.

[25] ——, "Apache Spark," https://spark.apache.org/.

[26] M. Dayarathna, S. Janakan, and R. Meertens, "Scaling a distributed stream processor in a containerized environment," InfoQ, Apr. 2019, https://www.infoq.com/articles/distributed-stream-processor-container/.

[27] Y. Guo, H. Shan, S. Huang, K. Hwang, J. Fan, and Z. Yu, "GML: Efficiently auto-tuning Flink's configurations via guided machine learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 12, Dec. 2021.

[28] A. Noureddine, R. Rouvoy, and L. Seinturier, "A review of energy measurement approaches," *Operating Systems Review*, vol. 47, no. 3, Dec. 2013.

[29] V. Weaver, "Reading RAPL energy measurements from Linux," https://web.eece.maine.edu/~vweaver/projects/rapl/.

[30] M. Colmant, R. Rouvoy, M. Kurpicz, A. Sobe, P. Felber, and L. Seinturier, "The next 700 CPU power models," *Journal of Systems and Software*, vol. 144, Oct. 2018.

[31] T. De Matteis and G. Mencagli, "Keep calm and react with foresight: Strategies for low-latency and energy-efficient elastic data stream processing," *ACM SIGPLAN Notices*, vol. 51, no. 8, Nov. 2016.

[32] C. Eibel, C. Gulden, W. Schröder-Preikschat, and T. Distler, "STROME: Energy-aware data-stream processing," in *Proc. DAIS*, 2018.

[33] S. Maroulis, N. Zacheilas, and V. Kalogeraki, "ExpREsS: EneRgy Efficient Scheduling of mixed stream and batch processing workloads," in *Proc. ICAC*, Jul. 2017.

[34] The Apache Software Foundation, "Apache Zookeeper," https://zookeeper.apache.org/.

[35] Cloud Native Computing Foundation, "Prometheus – monitoring system & time series database," https://prometheus.io/.

[36] Red Hat, "Ansible is simple IT automation," https://www.ansible.com/.

[37] The Linux Foundation, "Kubernetes," https://kubernetes.io/.

[1] https://gitlab.inria.fr/gkovilkk/greenflow

[38] J. von Kistowski, H. Block, J. Beckett, C. Spradling, K.-D. Lange, and S. Kounev, "Variations in CPU power consumption," in *Proc. ACM/SPEC ICPE*, 2016.
[39] S. Valstar, W. G. Griswold, and L. Porter, "Using DevContainers to standardize student development environments: An experience report," in *Proc. ACM Conference on Innovation and Technology in Computer Science Education*, Jun. 2020.
[40] D. Holtmann-Rice, S. Guadarrama, and N. Silberman, "Gin Config: a lightweight configuration framework for Python," https://github.com/google/gin-config.
[41] Python Software Foundation, "TinyDB," https://pypi.org/project/tinydb/.
[42] VictoriaMetrics, "VictoriaMetrics: Simple & reliable monitoring for everyone," https://victoriametrics.com/.
[43] The Linux Foundation, "Querying Prometheus," https://prometheus.io/docs/prometheus/latest/querying/basics/.
[44] Grafana Labs, "Grafana: the open observability platform," https://grafana.com/.
[45] R.-A. Cherrueau, M. Delavergne, A. van Kempen, A. Lebre, D. Pertin, J. Rojas Balderrama, A. Simonet, and M. Simonin, "EnosLib: A library for experiment-driven research in distributed computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, Jun. 2022.
[46] H. Arkian, G. Pierre, J. Tordsson, and E. Elmroth, "Model-based stream processing auto-scaling in geo-distributed environments," in *Proc. ICCCN*, Jul. 2021.