Adaptable Teastore with Energy Consumption Awareness: A Case Study

Henrique De Medeiros*(1,2), Denisse Muñante (1,3), Sophie Chabridon (1,2)
César Perdigão Batista (1,2), Denis Conan (1,2)
(1) SAMOVAR
(2) Télécom SudParis, Institut Polytechnique de Paris, 91120 Palaiseau, France
(3) ensIIE, Évry, France

Global energy consumption has been steadily increasing in the recent years, with data centres emerging as major contributors. This growth is largely driven by the widespread migration of applications to the Cloud, alongside a rising number of users consuming digital content. One way for reducing the energy consumption of data centres involves improving the energy efficiency of the deployed software applications. This implies design time and runtime approaches with self-adaptive applications. Although various strategies have been proposed, there is limited research comparing the architectural tactics employed in self-adaptive applications. In particular, there is a lack of evaluation regarding the energy consumption impact of architectural variants proposed in Cloud applications.

To address this gap, we propose a framework to integrate variants in the implementation of applications and allow to choose one particular variant according to its associated energy consumption at runtime. We add a monitor component to a Cloud application to track the energy usage of application containers and select adaptations according to the energy consumption of a microservice.

We have performed an empirical study on the Adaptable TeaStore application in a scenario of workload traffic on a node within the Grid5000 testbed. We assess the energy consumption of variants of the recommendation algorithm proposed by the TeaStore application, and we propose a functional level adaptation by analysing the behaviour of the application. First results show some potential improvements in the energy efficiency of cloud applications at runtime.

1 Introduction

Global energy consumption has been steadily increasing, with data centres emerging as major contributors. This growth is largely driven by the widespread migration of applications and businesses to the Cloud, alongside a rising number of users consuming digital content. To understand the connection between application behaviour and energy use, two key concepts are essential: energy efficiency and energy awareness. Energy-efficient applications are designed to operate with minimal energy impact, while energy-aware applications are developed with a conscious understanding of their energy footprint.

One of the challenges in building energy-efficient applications lies in accessing their energy consumption data. In Cloud environments, where infrastructure is managed by providers and resources are delivered on a pay-per-use basis, developers often rely on metrics like CPU and memory usage to gauge performance. However, these metrics alone do not reflect energy consumption. For example, CPU usage is frequently used as a proxy, but it does not provide a direct measurement of the energy required to perform specific tasks.

This paper addresses the dynamic adaptation of Cloud applications based on decisions regarding their energy consumption. For this purpose, we need to analyse the design-time and run-time aspects

*Corresponding author:
henrique.de_medeiros@telecom-sudparis.eu

Submitted to:
WACA 2025

© H. De Medeiros et al.
This work is licensed under the
Creative Commons Attribution License.

of applications. Design-time analysis helps to identify the components and requirements that can be integrated into the system to enhance energy efficiency. Following this, run-time analysis enables the enforcement of energy-efficient behaviour in Cloud applications without compromising their availability, by leveraging real-time insights about the system and its operating environment.

We rely on the case study of the Adaptable TeaStore [2] application and consider different recommendation algorithms. The Adaptable TeaStore simulates an online tea shop and supports architectural adaptations, making it an ideal candidate for studying energy-aware design in practice. We extend the Adaptable TeaStore application with energy awareness. While, in the base application, the CPU load is monitored to determine resource consumption, we also monitor the energy consumption of the application with a software energy profiling tool. Initially, we determine the energy consumption of the different recommender algorithms. In a second time, when the energy consumption of the application increases, an adaptation decision may allow to change the recommendation algorithm at run-time.

This paper is organised as follows: Section 2 provides background information and defines the key concepts used throughout the study. Section 3 describes the proposed extension of the Adaptable Teastore application. Section 4 specifies research questions and the designed experiment to address these research questions. Section 5 presents and analyses the results obtained. Section 6 reviews recent related works in the field. Finally, Section 7 concludes the paper.

2 Background

This section introduces the tools used for monitoring the energy consumed by software applications. Next, we introduce the TeaStore application. Finally, we explain the Adaptable TeaStore which is the extension of the TeaStore application to support its re-configuration at run-time.

2.1 Energy Monitoring Tools

The energy consumption of software applications can be measured through various approaches, *i.e.,* hardware-based power meters, software-based power meters and combined approaches. Jay *et al.* [7] categorise these approaches into four main groups: external devices, intra-node devices, hardware sensors with software interfaces and power/energy software models.

In Cloud computing environments, the hardware-based methods often fall short, measuring the total energy consumption of a machine or a computing node without offering sufficient granularity to attribute energy consumption to individual applications or processes running within the shared infrastructure. To address this limitation, software power meters have emerged as a practical alternative. These tools do not require external hardware and estimate energy consumption through mathematical models. Most of them rely on existing hardware sensors and software interfaces, such as Intel's RAPL (Running Average Power Limit) for CPUs and Nvidia's NVML (NVIDIA Management Library) for GPUs [3].

Based on the previous interfaces and memory consumption indicators, other software tools estimate energy consumption at the process level. Notable examples of software power meters include Linux perf, Kepler[1] (targeted for Kubernetes environments), PyJoules[2], PowerAPI[3], and Scaphandre[4]. PyJoules, PowerAPI, and Scaphandre are particularly interesting for their ability to measure energy consumption at the level of detail required for improving the energy-awareness of software applications.

[1] `https://sustainable-computing.io/`
[2] `https://github.com/powerapi-ng/pyJoules`
[3] `https://powerapi.org/`
[4] `https://github.com/hubblo-org/scaphandre`

2.2 The TeaStore application

TeaStore [9] is a distributed microservice application that emulates an online store for tea and tea related utilities. Figure 1 depicts its architecture. It is composed of six microservices, one of them being the *registry* service.

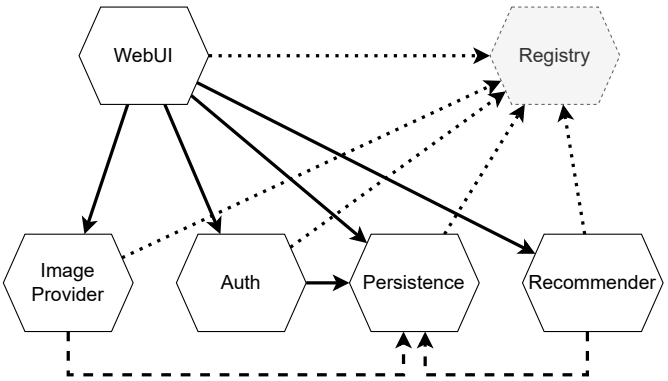


Figure 1: TeaStore Architecture, adapted from [9]

The WebUI service is the entrypoint for users' requests and serves the application web interface to support interactions. The Authentication service manages user login and session validation, thus ensuring secure access to the system. The Recommender service leverages rating algorithms to suggest products to users. Recommendations are generated based on the user behaviour, *e.g.,* the products being viewed by the user, the items in the shopping cart or the purchasing habits of other users. The Persistence service provides the information about the products and the user's purchases. The Image provider service is responsible for delivering the images used throughout the application. It dynamically adjusts image sizes based on the display context and incorporates caching mechanisms to ensure quick access to frequently requested images. The Registry service connects all active service instances by relying on their IP addresses or host-names and corresponding ports. It enables service discovery, thus allowing microservices to seamlessly locate and communicate with each other. All these services communicate using REST and Netflix Ribbon client-side load balancer.

2.3 The Adaptable TeaStore

Adaptable TeaStore [2] is an extension of the TeaStore application, and is designed to support dynamic adaptations when some particular events are triggered. Mandatory and optional services are identified (see Figure 2). WebUI, Persistence, and Image Provider correspond to the mandatory services, *i.e.,* they are required for all configurations of the TeaStore application, while Recommender and Authentication correspond to the optional services, *i.e.,* they could be present or not in a configuration of the TeaStore application. In addition, these optional services can also be adapted to adjust their resource consumption.

The Recommender service implements four recommendation algorithms, namely *Popularity*, *Slope One*, *Preprocessed Slope One*, and *Order-Based* methods. The algorithm is chosen at runtime, for instance, to alleviate/decrease the CPU usage.

After evaluating the source code of Adaptable TeaStore, we determine that only three potential reconfigurations are possible. The *Normal Mode*, or *Default Recommender*, restores the default recommendation algorithm, which is the *Slope One* algorithm. The *High Performance Mode* configures the

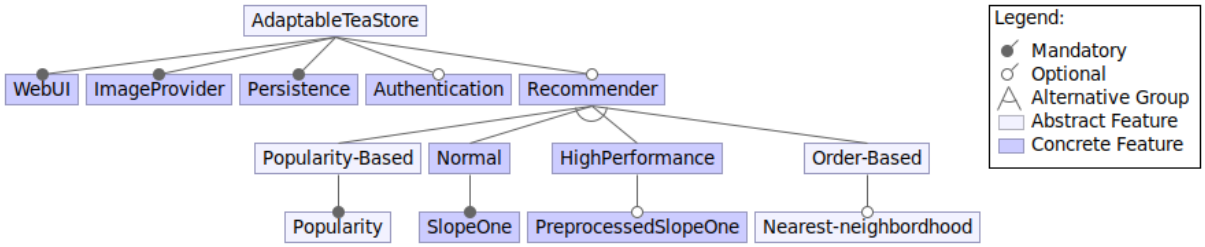


Figure 2: The feature model of Adaptable TeaStore

system to use the *Pre-processed Slope One* algorithm, aiming for maximum performance. And, the *Low Power Mode* disables the recommendation algorithm, *i.e.,* the recommender is deactivated. According to the source code, the popularity-based and the order-based algorithms are not enabled for adapting the Adaptable TeaStore. So in Figure 2, they are shown as abstract features that will not be included in the (re-)configurations of the Adaptable TeaStore.

Mode transitions are triggered based on CPU usage: when CPU usage exceeds 50%, the system switches to *Low Power* mode. If usage drops below 50%, it returns to *Normal* mode, *i.e.,* the Recommender is switched-on with the activation of the *Slope One* algorithm. Finally, in the current version of the Adaptable TeaStore, the *High Performance* mode is not activated for any event.

3 Extending Adaptable Teastore with Energy Consumption Awareness

To enable energy-awareness, we introduce the tool so-called EnCoMSAS (see the grey component in the top-right corner of Figure 3). This serves as a higher-layer energy consumption monitoring and analysis tool. It mainly allows to configure the monitoring frequency, enables the computation of the energy consumed by the target software application, and facilitates the analysis of energy consumption differences between different variants of the software application. The EnCoMSAS tool is designed not only for an specific microservice/application, *e.g.,* Adaptable TeaStore, but it is generic and can be used for any application. For this, EnCoMSAS receives the name of the software application, it then filters corresponding metrics, *e.g.,* power consumption, for this application, and it next computes energy consumption based on the previous metrics.

In this paper, EnCoMSAS uses Scaphandre for computing energy consumption, however EnCoMSAS is designed for being easily extended by adding another monitoring tool. Scaphandre reads instantaneous power consumption measurements of running applications, as well as of the entire machine, that are exposed with a Prometheus exporter. In order to evaluate the power consumption of particular microservices, the measurements should be filtered out to match the software application's (*e.g.,* Adaptable TeaStore) containers, which are mapped to specific PIDs. EnCoMSAS calculates energy consumption by using the equation $E = \int_{t_0}^{t_1} P(t)dt$, where t_0 is the timestamp at the start of the interval, t_1 is the timestamp at the end of the interval, and dt is the time increment over which power is integrated. The P(t) function is calculated by using a trapezoidal rule that increases the accuracy of the estimation of energy consumption over time. The choice of the rule is based on the approximation of integral E. The equation can be seen as follows: $E = \sum_{i=0}^{n-1}(\frac{P_i+P_{i+1}}{2} * \delta t)$, where P_i and P_{i+1} are successive power measurements at times t_i and t_{i+1}, $\delta t = t_i - t_{i+1}$ is the (constant) sampling interval, and n is the number of iterations for the energy consumption monitoring (so that $n+1$ samples yield n trapezoids, hence the summation index runs from $i = 0$ to $i = n-1$). Practically, EnCoMSAS is configured to calculate the energy consumption

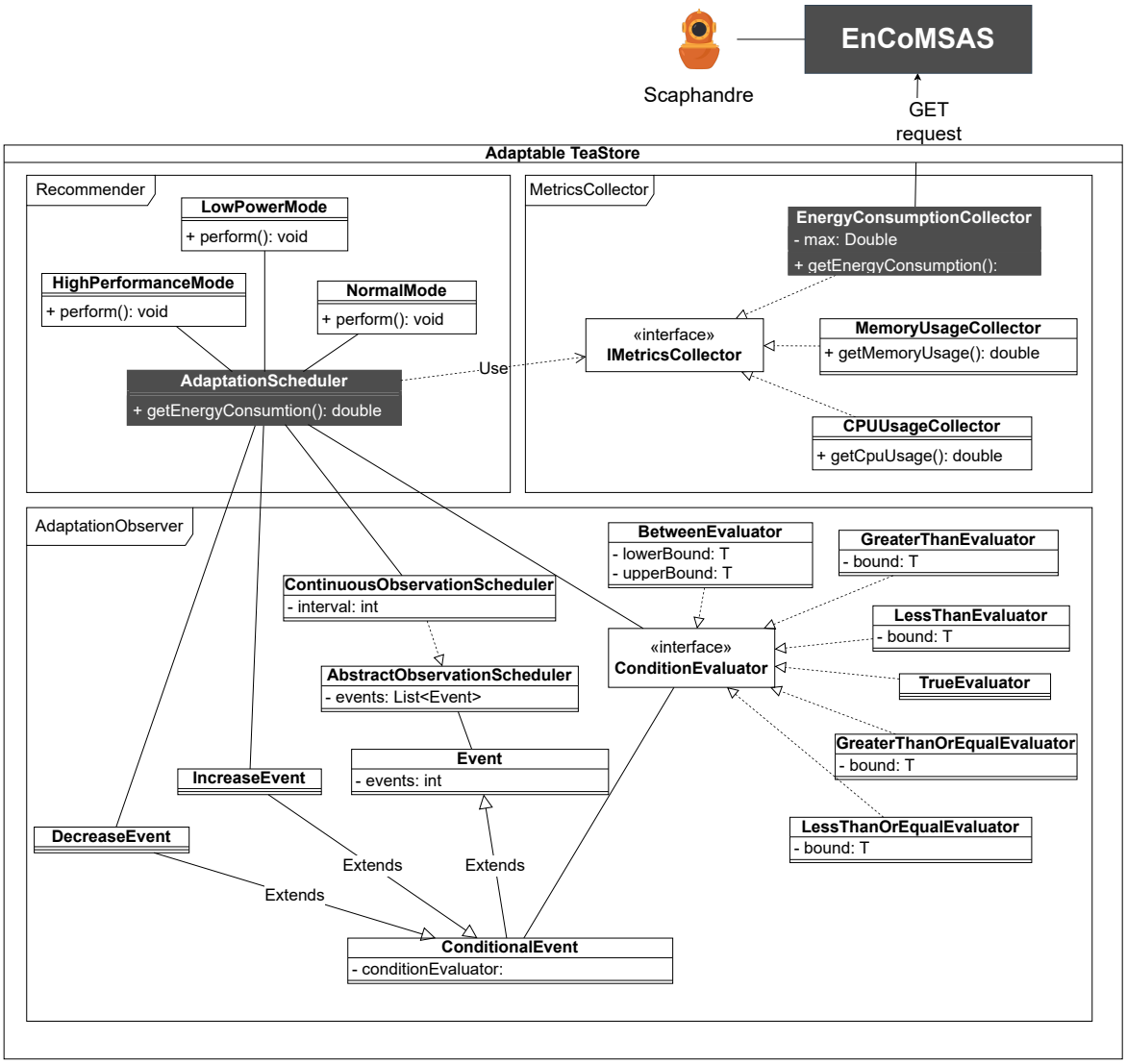


Figure 3: The extended architecture of Adaptable TeaStore

by intervals of 2 seconds. After executing the Scaphandre tool, we found that 2 seconds is the minimum time interval that allows to collect power consumption for calculating energy consumption. However, more experiments are needed to understand how this time interval could affect the energy consumption of the entire solution.

In order to enable energy awareness of Adaptable TeaStore, we extended its architecture as seen in Figure 3 (see the grey classes inside the Adaptable TeaStore component). The EnergyConsumptionCollector class is added and the AdaptationScheduler is updated to use the new collector. The EnergyConsumptionCollector class extends previous collectors by implementing the IMetricsCollector interface and by adding the getEnergyConsumption() method. This method collects the energy consumed through RESTful GET requests to an energy consumption monitor, *i.e.,* EnCoMSAS, that is instrumented by a power consumption profiling tool, namely Scaphandre. The `max` property serves to store the maximum

energy consumption obtained during the executions of the experiment; this value allows to normalise the energy consumption to make it comparable with other measurements.

The AdaptationScheduler class is introduced to allow the Recommender service to communicate with the energy consumption collector through the IMetricsCollector interface. It allows to gather energy consumption measurements when adaptations are performed.

The analysis performed in the AdaptationScheduler class leverages the existing structure provided in Adaptable TeaStore (for the CPU load analysis) by including the evaluation of energy consumption variations. The AdaptationScheduler class defines conditions required to trigger adaptations. These conditions are evaluated through the ConditionEvaluator interface, which provides several comparison operations, including: between two bounds, greater than, less than, etc. The conditions are periodically checked by the ContinuousObservationScheduler class: it sets up listeners for analysing the increase or decrease of the energy consumption at the application-level in order to ensure that adaptations are triggered as soon as the relevant conditions are met.

4 Experimentation Study

This section first presents the research questions and hypothesis for the study design. We then introduce the adaptation scenarios of the Adaptable TeaStore which serves as motivating examples for validating our proposal. Finally, we describe the protocol we followed for conducting a controlled experiment.

4.1 Research Questions, Hypothesis and Metrics

The goal of the study is to evaluate the effectiveness of EnCoMSAS to monitor energy consumption at design- and run-time. To do that, we run an experimental study in which EnCoMSAS is executed for enabling the comparison of the energy consumed by the different configurations of the Adaptable TeaStore. From this goal, we derive the following two research questions:

RQ1: Does EnCoMSAS enable the analysis of the impact of the variants of the Adaptable TeaStore on its energy consumption?

RQ2: Does EnCoMSAS enable the analysis of runtime adaptations of the Adaptable TeaStore on its energy consumption?

To answer these research questions, we conducted a controlled experiment. We identify as an **hypothesis** that "the service variants and adaptations at runtime influence the energy consumed by the Adaptable TeaStore, and the proposed EnCoMSAS should enable the evaluation of this influence". Moreover, the **metric** collected and analysed for this controlled experiment is focused on the energy consumption measurement (in Joules).

4.2 Adaptation scenarios of Adaptable TeaStore

In this paper, we focus on the recommender service of Adaptable TeaStore. Thus, we first configure Adaptable TeaStore for each of the four algorithms of the recommender service. Then, we gather the energy consumption of the five versions of Adaptable TeaStore, *i.e.,* one version for each algorithm and one version with a deactivated recommender service, and we provide statistical evidence for answering RQ1.

On the other hand, in [2], different scenarios of adaptation at runtime are introduced such as unavailable resources, cyber-attack, requirements changes or traffic increase. To answer RQ2, we focus on the *Benign Traffic Increase* scenario in which incoming traffic towards the WebUI increases significantly

due to a genuine increase in users' interactions. In order to handle the increased load, the application is adapted to the lower-power version by switching to a more energy-efficient recommendation algorithm. Thus, we investigate the impact on the energy consumed by Adaptable TeaStore under two distinct states: i) *without adaptations*, *i.e.,* using the default configuration (*Normal* mode) of the recommender service; and ii) *with adaptations*, *i.e.,* activating the *Lower Power* mode to use the most-energy efficient algorithm to reduce energy consumption.

4.3 Experimentation protocol

Figure 4 illustrates the execution flow of the experiment. We considered two execution environments: a local environment, where scripts run on a local machine, and the Grid5000 environment [4] [5].

To warm up the application, we deploy Adaptable TeaStore and run a medium-intensity workload, which is characterized by an increasing workload execution for 120 seconds, with maximum peak of 1000 requests in parallel during 10 seconds, as done in Kistowski *et al.* [9]. We then start the energy consumption monitor, that collects the energy consumed by the Adaptable TeaStore during 110 seconds. Once the energy consumption data is collected, it is exported into JSON files. Finally, the Grid5000 environment is cleaned up. This process is repeated 30 times. Once the number of iterations is finished, the analysis is performed to provide evidence about the extent to which the energy consumption was affected.

We use the protocol for two distinct scenarios: without and with adaptation. **Without adaptation** (NOADAPT), the application runs without changing its recommendation algorithm. These experiments allow to analyse each algorithm individually and define the most energy-efficient one. **With adaptations** (ADAPT), the application initially starts in the normal mode and dynamically transitions between high performance mode and the low power mode, differently from Adaptable TeaStore CPU analysis that disables the high performance mode and switches between the normal mode and the low power mode.

Finally, in our extension of Adaptable TeaStore, we added a flag to manage the activation of its

[5] `https://www.grid5000.fr/`

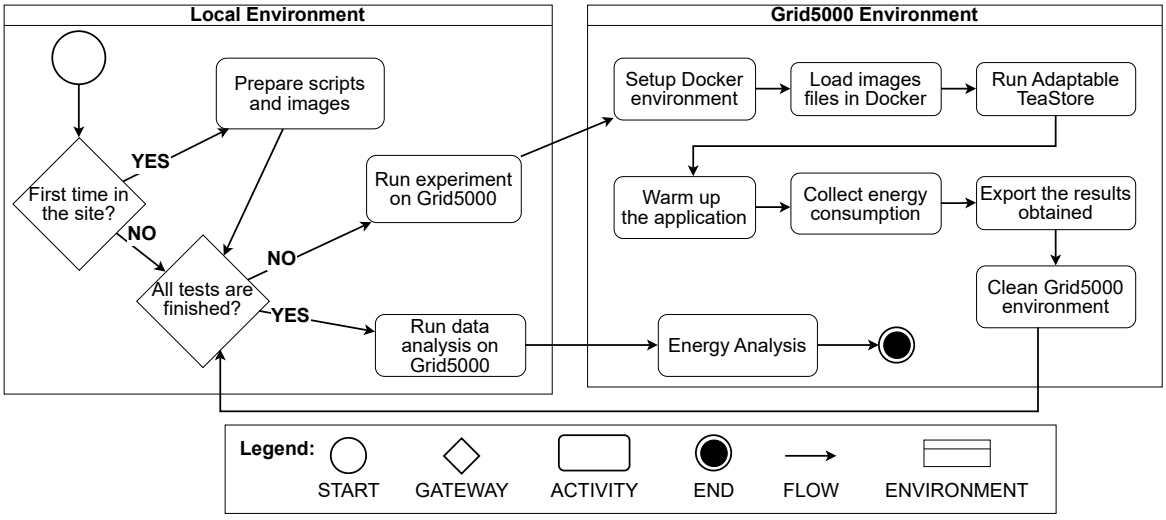


Figure 4: Step-by-step execution of the experiments

adaptable mechanisms. It allows to ensure that the execution of the NOADAPT scenario is not affected by the presence of adaptable components, specially the analysis component from Adaptable TeaStore that is permanently running.

5 Experimentation Results

To collect the data analysed in this section, the controlled experiment was conducted on the Nova cluster of the Grid5000 platform. The Nova cluster is composed of 20 nodes, each of these nodes is equipped with 64 GB of RAM, 2 CPUs Intel Xeon E5-2620 with 8 cores per CPU, 598 GB of storage with 25 GB allocated per user, and a 10 Gbps network connection.

As mentioned in Section 4.3, we executed 30 iterations of each scenario. To optimise the overall execution time of the experiment, the two scenarios (*i.e.,* with and without adaptation) were executed in parallel on two different nodes. Moreover, to simulate the user requests, we use the *HttpLoadGenerator* technique [8]. This technique is already implemented in Adaptable TeaStore and can be deployed in the Docker environment. Adaptable TeaStore proposes three workloads (*High*, *Medium* and *Low*). We activate the *Medium* workload to run the first scenario of the experiment and to answer RQ1, *i.e.,* study in which extent the recommender algorithms affect the energy consumption. Whilst for answering RQ2 by running the second scenario, *i.e.,* study how runtime adapatation (re-configurations) affect energy consumption, we first activate the *Medium* workload and increase it every two seconds, it is made by using the Load Generator implemented by the original TeaStore application that allows to increase quantity of activated users.

Finally, the workload for each scenario was always executed on the same node to ensure the use of the same computing resources. It allows keep the experiment controlled in order to gather energy consumption measurements that are statistically comparable.

5.1 RQ1: Does EnCoMSAS enable the analysis of the impact of the variants of the Adaptable TeaStore on its energy consumption?

To answer RQ1, we monitored Adaptable TeaStore after users requests were simulated and sent via an exposed API endpoint of the *WebUI service*. The *WebUI service* then calls the *Recommender service* that is already configured with the algorithm that is being evaluated. For the experiment, we study each of the four algorithms of Adaptable TeaStore and the version in which no recommender algorithm is activated.

Once the data are collected, we first analyze the distribution of the energy consumption measurements (in Joules). To do that, we used the `Shapiro` statistical test, which confirmed that the energy consumption collected from the 30 iterations executed for each version of Adaptable TeaStore presented a non-normal distribution curve. Next, we calculate the average values of the collected energy consumption measurements and calculate the relative standard deviation of the energy consumption over 30 iterations. These relative standard deviations show that the fluctuations of the values are small (i.e., less than 1) so the average values can be used as representative values for the 30 iterations. Finally, we perform the `Wilcoxon-Mann-Withney` statistical test to evaluate significance differences of energy consumption of the versions of Adaptable TeaStore.

Figure 5 and Table 1 (see Columns 2 and 3) present the distribution of the energy consumed by Adaptable TeaStore for the five versions (the deactivate version and the activation of each algorithm successively). As observed, the *Order-Based* version exhibits the highest outlier, so it presents the most significant deviation, *i.e.,* 28.59 as average with 1.1 as standard deviation, with respect to the others.

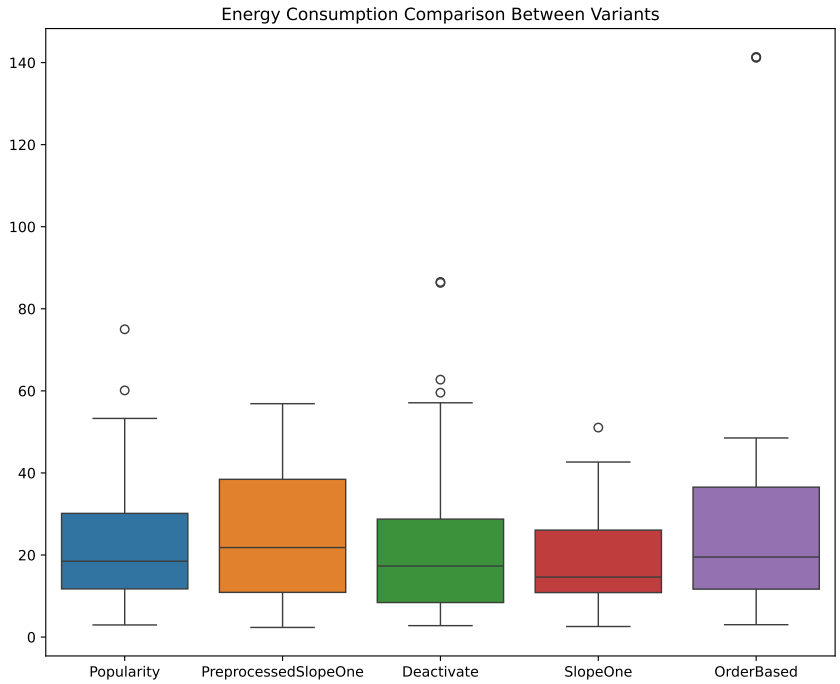


Figure 5: Distribution of the energy consumption

Table 1: The distribution of the energy consumed by the recommender algorithms, including the deactivating version. Where: *DEA = Deactivate*, *OB = Order-Based*, *POP = Popularity*, *SO = Slope One*, *PSO = Preprocessed Slope One*, *AEC = average of energy consumption (EC in Joule unit)*, *RSEC = relative standard deviation of EC*, *DEC = difference of EC vs Base*, *p-value= p-value of Wilcoxon test*

	AEC	**RSEC**	**DEA**		**SO**		**OB**		**POP**	
			%DEC	***p-value***	**%DEC**	***p-value***	**%DEC**	***p-value***	**%DEC**	***p-value***
DEA	18.10	0.64	–	–	–	–	–	–	–	–
SO	18.51	0.66	–	0.98	–	–	–	–	–	–
OB	**21.24**	**0.62**	–	0.18	–	0.13	–	–	–	–
POP	22.46	0.72	–	0.38	–	0.26	–	0.71	–	–
PSO	25.09	0.65	–	0.14	35.34	**0.046**	–	0.57	–	0.40

However, if we take out the outlier, the deviation is proved, *i.e.,* 21.24 as average with 0.62 as standard deviation (see Row 5 in Table 1). The *Popularity*, *Deactivate* and *Slope One* versions also present some outliers, however their distributions are comparatively closer to the average value of the distribution.

Moreover, *Deactivate* emerges as the most energy-efficient version with an average of 18.10J (and a relative standard deviation of 0.64). It is followed closely by the *Slope One* version with 18.51J as average (and 0.66 as relative standard deviation). The *Order-Based* version consumes 21.24J (and presents a relative standard deviation of 0.62, notice that these values were obtained after taking out the outlier value), the *Popularity* version consumes 22.46J and the *Preprocessed Slope One* version consumes the highest energy consumption at 25.09J, with relative standard deviation of 0.72 and 0.65, respectively.

As mentioned, we next analyse if there is any significant difference, *i.e.,* 95% of confidence or *p-value* <=5%, of the energy consumption of the executing versions of Adaptable TeaStore. To do that,

as the energy consumption data does not follow a normal distribution (it was confirmed through the `Shapiro` statistical test), we applied the `Wilcoxon-Mann-Whitney` statistical test. Table 1 summarises the results obtained from this statistical test. For instance, Columns 4 presents the significant percentages of difference of energy consumption between the *Deactivate* version versus the activation of each of the four recommender algorithms (notice – means not significant difference was obtained) ; and Column 5 introduces the *p-value* after executing the `Wilcoxon-Mann-Whitney` that helps deciding if difference is significant or not.

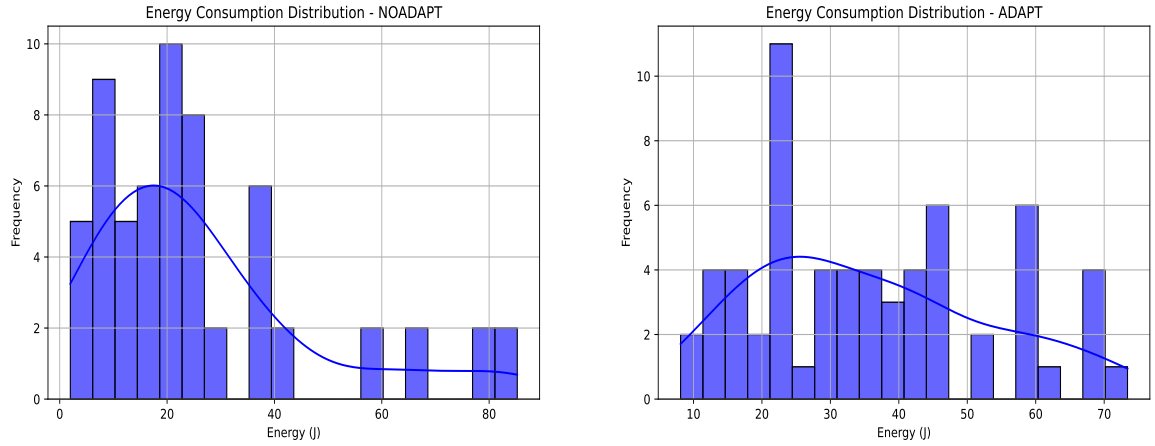
As observed in Table 1, the *Slope One* version and the *Preprocessed Slope One* version presents a significance difference of energy consumption of 35.34% with *p-value* of 0.046. It means that the *Preprocessed Slope One* version of Adaptable TeaStore consumes 35.34% more energy with respect to the *Slope One* version of Adaptable TeaStore. For the rest of comparisons no significant differences were obtained.

> **In response to RQ1:** Seeing the results, we demonstrated that EnCoMSAS enables the analysis of the impact on the energy consumption of the recommender service variants of Adaptable TeaStore. After running the controlled experiment, we found that the *Preprocessed Slope One* version consumes 35.34% more energy than the *Slope One* version. No more significant differences were found.

5.2 RQ2: Does EnCoMSAS enable the analysis of runtime adaptations of the Adaptable TeaStore on its energy consumption?

To answer RQ2, we conduct a controlled experiment following the protocol described in Section 4.3. The experiment was conducted under two scenarios: with adaptation (ADAPT) and without adaptation (NOADAPT). For collecting energy consumption measurements, we run Adaptable TeaStore instrumented with EnCoMSAS.

According to the feature model of Adaptable TeaStore introduced in Figure 2, the recommender service has two available modes, *i.e.,* Normal and High Performance modes that respectively activate the Slope One and the Preprocessed Slope One algorithms.



(a) Energy Consumption of NOADAPT

(b) Energy Consumption of ADAPT

Figure 6: Distribution of the energy consumption of two scenarios

As observed in Figure 6, ADAPT tends to consume more energy than NOADAPT. The difference in the energy consumption distribution is attributed to the adaptive behaviour of the Recommender service in the ADAPT scenario of the application. At runtime, the Recommender dynamically selects different algorithms based on system conditions, *i.e.,* reduce the energy consumption in response of an increasing workload and relax the energy consumption restriction when workload is reduced. This behaviour is particularly evident in the energy interval between $40J$ and $60J$, as shown in both graphs. In this range, the adapted version consistently demonstrates higher energy consumption, whereas the non-adaptive version shows a gradual decrease in energy usage.

Regarding the NOADAPT scenario, we chose the default configuration of the Adaptable TeaStore which is the Normal Mode. It means that the NOADAPT scenario runs the Slope one algorithm and no adaptations are executed.

Whilst for the ADAPT scenario, based on the results from the algorithm analysis in the previous section and the available adaptation mechanisms in the application, we configured the Adaptable TeaStore to use the Slope One algorithm, identified as the most energy efficient, for the Normal Mode. For the High Performance Mode, we retained the Preprocessed Slope One algorithm, as it is already implemented in the application's source code.

Figure 7 and Table 2 present the distribution of energy consumption across both scenarios. As observed, the lowest recorded energy consumption occurs in the non-adaptive scenario, while the highest consumption appears in the adaptive scenario. The ADAPT version present a energy consumption average of 45.44 and a RSEC of 0.72, while NOADAPT presents 32.53 for the average and the RSEC

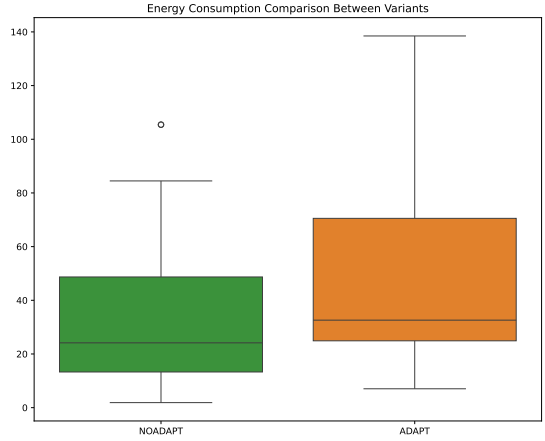


Figure 7: Data distribution of energy consumption between variants

Table 2: The distribution of the energy consumed by the Adaptable TeaStore versions. Where: *NOADAPT = No adaptation*, *ADAPT = Adaptable version*, *AEC = average of energy consumption (EC in Joule unit)*, *RSEC = relative standard deviation of EC*, *DEC = difference of EC vs Base*, *p-value= p-value of Wilcoxon test*

	AEC	**RSEC**	**ADAPT**	
			%DEC	***p-value***
ADAPT	45.44	0.72	–	–
NOADAPT	32.53	0.79	**39.69**	**0.011**

of 0.72. Based in the *p-value*, there is a significant difference between the both approaches, ADAPT consumes 39.69% more energy with respect to NOADAPT.

In the ADAPT version, we need to consider that the recommendation algorithms will learn from the workloads and provide the products according to this task, increasing the energy consumption.

Over time, the non-adaptive version exhibits a noticeable upward trend in energy consumption, occasionally surpassing the adapted version in terms of peak usage. This fluctuation results in alternating periods where the highest energy consumption is recorded in the non-adaptive scenario.

In response to RQ2: Seeing the results, we demonstrated that EnCoMSAS enables the analysis of energy consumption for the run-time adaptations of Adaptable TeaStore. After running the controlled experiment, we found that using the application with adaptation requires more energy consumption than the version without adaptation. Running the experiments, we got that the adaptation version consumes 39.69% more than the version without adaptation (with the most energy efficient algorithm).

6 Related Work

The related work can be broadly categorised into two areas: research focused on designing energy-efficient architectures that offer tactics and patterns for building greener Cloud applications, and research exploring runtime adaptations that consider energy-consumption metrics. The latter remains less explored, especially when considering performance-related metrics such as energy consumption rather than CPU load or memory usage.

Energy-efficient architectures. The energy consumption of Cloud applications has been widely explored across various fields in the literature. Some studies estimate energy usage based on the application's consumption of machine resources such as CPU and memory. Others take a broader approach by measuring the host machine's overall energy consumption to infer the energy footprint of the application.

In [15], Xiao *et al.* analyse tactics and patterns from the literature related to energy consumption in microservices, selecting three tactics and three patterns, and subsequently examining the trade-offs with maintainability and performance. In [11], Noureddine and Le Goaer follow the same principle, analysing the impact of desgin patterns on energy consumption. In contrast, our work presents an in-depth analysis of adaptation at the functional level of a microservice.

In [14], Werner *et al.* propose a Kubernetes-based framework for evaluating Cloud applications using sustainability, quality, and performance metrics across different architectural layers. Their analysis offers a high-level view of application metrics. In contrast, our work provides a more fine-grained, function-level analysis focused solely on energy consumption. We extended a Cloud application that supports adaptations based on CPU usage to also support adaptations triggered by energy consumption at runtime, offering deeper insight beyond the broader design-time architectural analysis presented in their study.

Run-time Application Adaptations. Existing self-adaptive systems that address energy consumption typically base decisions on host-level resource usage. For instance, in [10], Malik *et al.* aim to reduce energy consumption by scaling resources based on performance metrics and power meter readings. Berkane *et al.* propose in [1] some elastic scaling rules informed by user activity and server utilisation, while Huber *et al.* [5] focus on minimising resource usage while preserving response time.

Our work differs by targeting energy consumption at the application level, independently of infrastructure metrics. This approach allows us to isolate and understand the impact of runtime adaptations driven solely by energy usage, particularly through the selection of algorithms designed to optimise energy efficiency of one microservice.

7 Conclusion

Understanding the energy consumption of applications is crucial for developing energy-efficient solutions. Through the analysis, developers can make better decisions about which architectural adaptations best suit specific scenarios. For instance, in an online store environment, like in this work, periods of high demand such as during peak sales seasons may call for performance-optimised configurations, even at the cost of increased energy usage. By balancing performance requirements with energy-consumption considerations, it becomes possible to create adaptable applications that are both energy-efficient and responsive to changing conditions. Therefore, in this paper, we introduced the EnCoMSAS tool that allows to compute and analyse the energy consumed by software applications at design- and runtime, and in no adapted and dynamic adapted scenarios. Moreover, EnCoMSAS is able to set up the monitoring frequency.

Based on the conducted experiments to evaluate the effectiveness of EnCoMSAS, we first assess the extent in which the algorithms implemented by Adaptable TeaStore for its recommender service affect the energy consumption. We found that the most energy efficient algorithm is Slope One. A second controlled experiment was performed to evaluate the extent in which the adaptation version of Adaptable TeaStore affect its energy consumption with respect to its non adaptation version. It proofs the effectiveness of EnCoMSAS for running this experiments in which the response time is required. However, more experiments are needed in order to understand what is the impact of EnCoMSAS on qualitiy of services attributes of software applications such as performance and energy efficiency itself.

Another future work is to study what is the impact to have an external component, *i.e.,* not as part of the Adaptable TeaStore, for the adaptations analysis and planning, such this is made in the MAPE-K loop [6]. Centralising the adaptation logic in a single internal service, as observed in our experiment, may limit the effectiveness of the energy efficiency. By decoupling adaptation decisions from the core services, it becomes possible to make more informed and flexible energy-aware adjustments across the application ecosystem. However, we should run more experiments to confirm our hypothesis.

Acknowledgements

The authors thank Yakine Klabi and Sara Kossentini for their work on the study of the variants of the recommendation service for Adaptable TeaStore: it was made during their Master research project at Télécom SudParis. This research was produced within the framework of Energy4Climate Interdisciplinary Center (E4C) of IP Paris and Ecole des Ponts ParisTech. This research was supported by 3rd Programme d'Investissements d'Avenir [ANR-18-EUR-0006-02]. This work received funding from the France 2030 program, managed by the French National Research Agency under grant agreement No. ANR-23-PECL-0003.

References

[1] Mohamed Lamine Berkane, Mahmoud Boufaida & Nour El Houda Bouzerzour (2022): *Modelling elastic scaling of cloud with energy-efficiency: Application to smart-university*. *Journal of King Saud University - Computer and Information Sciences* 34(6, Part B), pp. 3136–3150, doi:https://doi.org/10.1016/j.jksuci.2020.11.025. Available at `https://www.sciencedirect.com/science/article/pii/S1319157820305541`.

[2] Simon Bliudze, Giuseppe De Palma, Saverio Giallorenzo, Ivan Lanese, Gianluigi Zavattaro & Brice Arleon Zemtsop Ndadji (2024): *Adaptable TeaStore*. arXiv:2412.16060.

[3] Emile Cadorel & Dimitri Saingre (2024): *A Protocol to Assess the Accuracy of Process-Level Power Models*. In: *2024 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 74–84, doi:10.1109/CLUSTER59578.2024.00014.

[4] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier & O. Richard (2005): *Grid'5000: a large scale and highly reconfigurable grid experimental testbed*. In: *The 6th IEEE/ACM International Workshop on Grid Computing, 2005.*, pp. 8 pp.–, doi:10.1109/GRID.2005.1542730.

[5] Nikolaus Huber, Fabian Brosig & Samuel Kounev (2011): *Model-based self-adaptive resource allocation in virtualized environments*. In: *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '11, Association for Computing Machinery, New York, NY, USA, p. 90–99, doi:10.1145/1988008.1988021. Available at `https://doi.org/10.1145/1988008.1988021`.

[6] IBM (2005): *An Architectural Blueprint for Autonomic Computing*. Technical Report, IBM.

[7] Mathilde Jay, Vladimir Ostapenco, Laurent Lefevre, Denis Trystram, Anne-Cécile Orgerie & Benjamin Fichel (2023): *An experimental comparison of software-based power meters: focus on CPU and GPU*. In: *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 106–118, doi:10.1109/CCGrid57682.2023.00020.

[8] Jóakim von Kistowski, Maximilian Deffner & Samuel Kounev (2018): *Run-time Prediction of Power Consumption for Component Deployments*. In: *Proceedings of the 15th IEEE International Conference on Autonomic Computing (ICAC 2018)*.

[9] Jóakim von Kistowski, Simon Eismann, Norbert Schmitt, André Bauer, Johannes Grohmann & Samuel Kounev (2018): *TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research*. In: *Proceedings of the 26th IEEE International Symposium on the Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, MASCOTS '18, doi:10.1109/MASCOTS.2018.00030.

[10] Maria Malik, Hassan Ghasemzadeh, Tinoosh Mohsenin, Rosario Cammarota, Liang Zhao, Avesta Sasan, Houman Homayoun & Setareh Rafatirad (2019): *ECoST: Energy-Efficient Co-Locating and Self-Tuning MapReduce Applications*. In: *Proceedings of the 48th International Conference on Parallel Processing*, ICPP '19, Association for Computing Machinery, New York, NY, USA, doi:10.1145/3337821.3337834. Available at `https://doi.org/10.1145/3337821.3337834`.

[11] Adel Noureddine & Olivier Le Goaer (2025): *Investigating the Impact of Software Design Patterns on Energy Consumption*. In: *22nd IEEE International Conference on Software Architecture*, Odense, Denmark. Available at `https://hal.science/hal-04843913`.

[12] Carlos V Paradis, Rick Kazman & Damian Andrew Tamburri (2021): *Architectural Tactics for Energy Efficiency: Review of the Literature and Research Roadmap*. In: *HICSS*, pp. 1–10.

[13] András Vargha & Harold D. Delaney (2000): *A critique and improvement of the CL Common Language effect size statistics of McGraw and Wong*. *Journal of Educational and Behavioral Statistics* 25(2), pp. 101–132, doi:10.3102/10769986025002101.

[14] Sebastian Werner, Maria C. Borges, Karl Wolf & Stefan Tai (2025): *A Comprehensive Experimentation Framework for Energy-Efficient Design of Cloud-Native Applications*. arXiv:2503.08641.

[15] Xingwen Xiao, Chushu Gao & Justus Bogner (2025): *On the Effectiveness of Microservices Tactics and Patterns to Reduce Energy Consumption: An Experimental Study on Trade-Offs*. arXiv:2501.14402.