2024 IEEE 21st International Conference on Software Architecture (ICSA)

Exploring sustainable alternatives for the deployment of microservices architectures in the cloud

Vittorio Cortellessa
SPENCER Lab
University of L'Aquila, Italy
vittorio.cortellessa@univaq.it

Daniele Di Pompeo
SPENCER Lab
University of L'Aquila, Italy
daniele.dipompeo@univaq.it

Michele Tucci
SPENCER Lab
University of L'Aquila, Italy
michele.tucci@univaq.it

2024 IEEE 21st International Conference on Software Architecture (ICSA) | 979-8-3503-5916-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICSA59870.2024.00012

***Abstract*—As organizations increasingly migrate their applications to the cloud, the optimization of microservices architectures becomes imperative for achieving sustainability goals. Nonetheless, sustainable deployments may increase costs and deteriorate performance, thus the identification of optimal trade-offs among these conflicting requirements is a key objective not easy to achieve. This paper introduces a novel approach to support cloud deployment of microservices architectures by targeting optimal combinations of application performance, deployment costs, and power consumption. By leveraging genetic algorithms, specifically NSGA-II, we automate the generation of alternative architectural deployments. The results demonstrate the potential of our approach through a comprehensive assessment of the Train Ticket case study.**

***Index Terms*—sustainability, refactoring, performance, search-based software engineering, model-driven engineering**

I. Introduction

As the world's reliance on digital technologies grows, so does the environmental impact of data centers. The escalating demands for computing resources, coupled with the energy-intensive operations of data centers, significantly contribute to their substantial carbon footprint [1]. Cloud computing is at the center of these demands, as it has emerged as a key solution to enhance technological capabilities of companies. The prevailing trend in cloud deployment involves designing applications based on the principles of microservices architecture [2]. This architectural approach is widely favored for cloud environments, as it aligns seamlessly with the dynamic and scalable nature of cloud computing [3].

When deploying microservices applications in the cloud, the energy footprint of the application is frequently overlooked, whereas prioritizing concerns such as performance and deployment costs, which are more closely related to operational success [4]. The challenge of considering energy consumption stems from the introduction of an additional dimension into deployment planning, and from the inherent complexity in evaluating the energy requirements across various potential alternative configurations [5]–[7]. Nonetheless, nowadays it is essential to consider sustainability as a trade-off of performance, deployment cost, and energy consumption [8] to ensure the long-term viability of cloud-based microservices architectures.

A number of approaches [9]–[11] emerged in recent years to optimize energy consumption and cost when deploying to the cloud. Nevertheless, these approaches seldom address this problem at the architectural level and, as a consequence, often they lack the capability to empower designers with a comprehensive understanding of the intricate trade-offs emerging from this task. This lack of understanding is further exacerbated by the fact that the energy consumption of a microservices architecture is not only a function of the deployment configuration but also of the user behavior [12].

In this paper, we aim to address this lack by presenting a novel approach to explore sustainable solutions when deploying microservices architectures in the cloud. Specifically, we exploit NSGA-II [13] to generate diverse deployment configurations through the application of refactoring actions to an initial architecture. Our method provides the designer with design alternatives that represent optimal trade-offs of system performance, deployment costs, and power consumption. Moreover, we support the exploration of the intricate relationship between power consumption, cost and user behavior by observing the distribution of these factors on different types of user requests. Finally, we investigate how architectural solutions change when optimizing for power consumption, thus aiming to discern recurring refactoring actions in a power-aware context.

To enhance the practical significance of our research, we showcase our approach on the Train Ticket Booking Service case study [3], which is a widely used microservices benchmark designed to reflect a real-world scenario.

Our results reveal that the introduction of a power consumption objective significantly affects system response time and, contrary to conventional expectations, has a negligible impact on deployment costs. Also, the contribution of individual types of requests greatly varies depending on the deployment decisions. In perspective, our approach can identify opportunities for merging smaller microservices to save on power consumption and costs.

The remainder of this paper is organized as follows. Sec-

2835-7043/24/$31.00 ©2024 IEEE
DOI 10.1109/ICSA59870.2024.00012

tion II presents the proposed approach. Section III describes our settings and experiments. Sections IV to VI discuss the research questions and present the results. Section VII presents the related work. Section VIII presents the threats to validity. Section IX concludes the paper and discusses future work.

II. Approach

Our approach is based on genetic algorithms, which are bio-inspired algorithms that achieve the desired near-optimal Pareto fronts by evolving the specie through crossover and mutation operators. The specie in our context is the set of candidate architectural solutions generated through refactoring, and the genetic algorithm is NSGA-II [13]. NSGA-II is widely used in the literature and it is considered one of the most effective multi-objective evolutionary algorithms [14]. Its main characteristic is the use of a fast non-dominated sorting approach to select the best individuals in the population. As a result, NSGA-II is able to achieve a good convergence and diversity of the Pareto front.

An individual in our population is a sequence of refactoring actions. Each action is applied to an element in the software architecture. When combining refactoring actions to compose the sequence, the algorithm must respect constraints (i.e., pre- and post-conditions) that are specific to each action. Therefore, once an individual has been created, it can be applied to the architecture to generate a new architectural alternative.

A. Objectives

The goal of this approach is to generate alternative architectures with lower power consumption, while preserving, or even improving, the performance, by minimizing the overall economic cost of deployment and the refactoring effort to change the architecture. In the following, we describe the objectives that are considered in this study.

a) Power consumption: Xu and Jian [15] proposed a model to estimate the total power consumption of a server in case its power consumption is only known when it is fully utilized. By considering that most of the power consumption of a server comes from its CPU [16]–[18], Xu and Jian in their model compute the power consumption of a server as the combination of power consumption of the CPU when used and when idle, where the latter one is obtained by scaling down the former by a factor.

We adopt their model, thus we estimate the power consumption of an application as the sum of the power consumption of all the nodes (i.e., servers on which it is deployed), as follows:

$$power = \sum_{n \in N} (1 - U_n) \cdot k \cdot power_{max}(n) + U_n \cdot power_{max}(n)$$

where n is a node among the N used ones, U_n is its utilization, $power_{max}(n)$ is the maximum power consumed when n is fully utilized, k is the scaling factor, introduced by Xu and Jian, by which $power_{max}$ is reduced when the server is idle.

b) Response Time: Performance of a software system is a broad term that can be quantified by several metrics. In this study, we consider system response time as the performance metric to be minimized. It can be obtained as the combination of response times of the system when it is triggered by different types of requests. This metric is a common outcome of tools that solve performance models.

c) Complexity: Refactoring actions can be more or less complex to be applied on a software architecture. In this study, we introduce a complexity metric defined as follows:

$$complexity = \sum_{a=1}^{l} C_{base}(a) \cdot C_{arch}(a, e)$$

where $C_{base}(a)$ is the base complexity associated to a type of action a, and $C_{arch}(a, e)$ is the architectural complexity that is computed on the basis of the element e on which the refactoring is applied, as explained in the following. C_{base} can be estimated through several estimation models [19], [20]. Often, these models exploit knowledge of the software system and implement business rules that are hard to generalize. For this reason, we estimated the base complexity of each type of action on the basis of prior studies on software architecture refactoring [21], [22]. The $C_{arch}(a, e)$ complexity, instead, is related to the target element of the refactoring action, because a type of action can have different complexities when applied to different target elements of the architecture. For example, the more interconnected is a target element, the higher is the architectural complexity of the refactoring action. Or, in other words, target elements that are used relatively less than other ones induce a lower complexity, due to the lighter impact that they have on the rest of the architecture.

d) Cost: The deployment cost of a software architecture in a cloud based infrastructure is the cost of the hardware nodes on which it is deployed. Thus, it is related to the number and the type of nodes used to deploy the architecture.

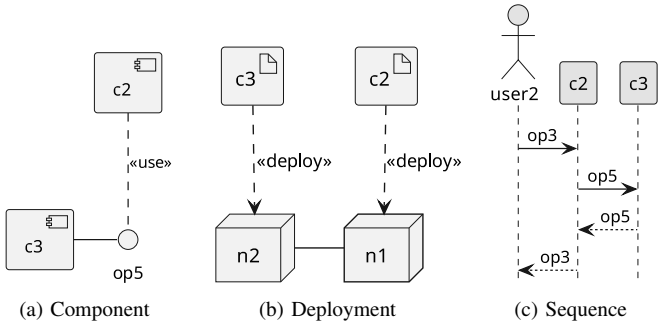
Hence, we considered the overall cost of the software architecture as the sum of the hourly cost of all the used nodes (N), as follows:

$$cost = \sum_{n \in N} cost(n)$$

B. Modeling assumptions

In this study, we use UML to build a software architecture of a system. In particular, we use a Component Diagram to model static connections among software components, Sequence Diagrams to model the dynamic behavior of the software system, and Deployment Diagram to model the hardware platform and the component allocation.

Figure 1 shows simple examples of UML diagrams used in this paper. Natively, UML does not provide notation elements to model performance, but the MARTE profile [23] has been introduced for this goal. In this study, we use Layered Queueing Networks (LQN) [24] to model and analyze the performance of the software architecture. Several approaches have been introduced to transform a UML model into a



(a) Component (b) Deployment (c) Sequence

Fig. 1. UML diagrams

performance model [25]–[27], and here we adopt the approach in [28] to obtain the LQN model.

C. Refactoring catalog

In this study, the refactoring catalog is made up of refactoring actions that were conceived to improve software performance. We use the five types of refactoring actions described in the following.

a) REDO - Redeploy Existing Component: Redeploying an existing component involves modifying the deployment by relocating a selected component to a newly created node. This action aims to optimize the software architecture by strategically redistributing components, while ensuring that the new node has connections with all nodes directly linked to the original one.

b) MOVE - Relocate Operation to Existing Component: The relocation of an operation to an existing component is an action that involves selecting and transferring a specific operation to an existing target component. This action propagates modifications in all scenarios where the operation occurs.

c) CLON - Clone Node: Cloning a node is an action aimed at introducing a replica of an existing node. It ensures that every deployed component and connection of the original node is duplicated, thus also contributing to system redundancy and fault tolerance.

d) MOTN - Move Operation to New Component on New Node: Moving an existing operation to a new component on a new node is a complex refactoring action that requires consistency of static, dynamic and deployment views. This involves adding the newly created component in the dynamic view and creating a new node, artifact, and related links in the deployment view. Messages addressed to the moved operation are appropriately forwarded, thus ensuring the seamless integration of the new component.

e) DROP - Remove Node: A node removing action induces the relocation of its deployed components on neighbor nodes, in order not to remotely spreading components that could be highly interconnected.

III. Experimental setup

In this section, we describe the experimental setup used to evaluate the proposed approach.

A. Research questions

This study aims to answer the following research questions.

RQ1. *What is the impact of sustainability on performance and cost?*
Rationale: Adding sustainability constraints can negatively impact performance and cost of a software system. We aim at estimating what we need to trade for more sustainable deployments.

RQ2. *What is the effect of refactoring actions on the distribution of power and cost across different types of requests?*
Rationale: The refactoring actions can change the power and cost distribution across different user behaviors (i.e., different types of requests). We observe these distributions in optimal solutions, thus providing a view that may enable architectural decisions depending on user behaviors.

RQ3. *How do the architectural solutions change when introducing power consumption among the optimization objectives?*
Rationale: The introduction of power consumption among objectives may lead to use different refactoring actions. Here, we aim at studying their frequencies to identify possible recurring choices.

B. Case study

We experimented our optimization model on a software system, namely Train Ticket Booking Service (TTBS), which is a web-based booking application whose architecture is based on the microservices paradigm [3]. The system is made up of 40 microservices, each one deployed on a Docker container. Among all the TTBS scenarios that a user can trigger, we focused on the most common and critical ones. In particular, we consider the following scenarios: *Login*, *Update user details*, and *Rebook a ticket*.

The architectural specification, including component, deployment, and sequence diagrams (in UML format), and the analytical models (LQN) of TTBS are provided for experiment replication[1].

C. Experimental design

a) Setting the model: First, we built a UML model of the TTBS. Then, we augmented the model with the MARTE profile [23]. In particular, we used the *GQAM*, *HwLayout*, and *GRM* packages to specify the performance, power, and cost properties, respectively.

With regard to performance properties, we gathered data from a running TTBS application to set performance input data (*e.g.*, operations demand vectors), as suggested in [29], [30]. Furthermore, we compute the speed factor of an instance as the PassMark average CPU mark of the used CPU[2] scaled over the number of virtual CPUs provided by the Amazon EC2 instance type.

[1]Dataset: https://zenodo.org/doi/10.5281/zenodo.10246197
[2]PassMark Software CPU Benchmark: https://www.cpubenchmark.net/

TABLE I
INSTANCE TYPES AND DESCRIPTIONS. EACH AMAZON EC2 INSTANCE TYPE HAS BEEN LABELED WITH RESPECT TO *Speed Factor*, *Power Consumption* IN WATT, AND *Cost* IN USD/H.

Instance Type	Description	Speed Factor	Power Consumption	Cost
`d2.2xlarge`	Designed for resource-intensive workloads like scientific simulations, machine learning, or real-time analytics.	4.67	83.4	0.46
`m6i.xlarge`	Offers good performance for demanding web applications, database servers, or content delivery with lower energy consumption and operational cost.	3.48	32.4	0.13
`t2.medium`	Well-balanced for a wide range of general-purpose workloads, cost-effective for typical business applications, small to medium-sized websites, and development environments.	2.33	14.1	0.03
`t2.micro`	Best suited for non-resource-intensive tasks like simple web hosting, low-traffic blogs, or small-scale personal projects with low energy consumption and operational cost.	1.17	6.40	0.004
`m5ad.xlarge`	Suitable for tasks where a balance between energy efficiency and operational cost is desired, such as basic web hosting or non-resource-intensive background tasks.	1.14	29.9	0.25

To estimate the power consumption under full utilization of CPU (i.e., $power_{max}(n)$), we extracted data from the Amazon EC2 Instances Carbon Footprint Estimator dataset[3]. From the same dataset, we have estimated the k factor that we adopt to scale down the power when the CPU is idle.

Moreover, we set the cost of each instance by using the information provided in the Amazon EC2 Price History dataset[4].

We selected five Amazon EC2 instances among the all the available ones, namely *d2.2xlarge*, *m5ad.xlarge*, *t2.medium*, *t2.micro* listed in Table I, as they represent conflicting trade-offs. The selected instances have balanced characteristics in terms of performance, power consumption, and cost. Thus, we avoid the algorithm to have a strong preference when searching the solution space. In other terms, if an instance has the best value for each characteristic, the algorithm will always select it when searching the solution space because that instance will improve all the considered objectives.

b) Setting the algorithm: Once the source model has been built, we set up the optimization problem by defining objectives and constraints. We set the objective function to minimize power consumption, response time, cost, and complexity. We use the NSGA-II algorithm to build the Pareto front of the optimization problem.

We performed several trials to find the best configuration of the optimization problem. Finally, we set the size of the initial population of the NSGA-II algorithm to 16 individuals, the maximum number of generations to 200, and crossover and mutation probabilities to 0.8 and 0.2, respectively. In our problem, the chromosome represents the sequence of refactoring actions that the designer can perform on the system. We decided to limit this sequence length to four refactoring actions, because it seemed a reasonable number of actions that a designer can perform in sequence without leading the system architecture too far from the initial one.

To mitigate the randomness of the optimization algorithm, we run the optimization process 31 times for each scenario, as suggested by Arcuri and Fraser [31].

c) Setting the experiments: In order to answer the research questions, we performed two different types of experiments, namely *baseline* and *power-aware*. The *baseline* experiment does not consider power consumption as an objective of the optimization problem while keeping the other three (*i.e.*, response time, complexity, cost), whereas the *power-aware* one adds the power consumption to the objectives.

It is worth to note that, for sake of full comparison, the power consumption has been computed also for the solutions obtained in the *baseline*, even though it has not been considered as an objective of the optimization problem.

IV. WHAT IS THE IMPACT OF SUSTAINABILITY ON PERFORMANCE AND COST? (RQ1)

By comparing a *baseline* experiment with one that is *power-aware*, we aim at investigating how the objectives in our approach vary when power consumption is considered as an additional objective. Clearly, we expect that, on average, the values of the power objective will be lower in the *power-aware* experiment because power is explicitly targeted by the optimization algorithm. Nonetheless, it is important to understand how the power objective affects the other ones and, more concretely, how much we could trade in performance and cost for more sustainable solutions. In this context, we still kept the *complexity* objective to avoid considering disruptive sequences of refactoring actions on the initial architecture.

A. Differences in the Pareto fronts

Figure 2 shows the Pareto fronts obtained in the *baseline* and *power-aware* experiments, which are obtained as the super-Pareto front of the 31 runs of each experiment. Namely, each solution in a super Pareto front is not dominated by any other solution in any of the 31 runs.

[3]Amazon EC2 Instances Carbon Footprint Estimator: https://docs.google.com/spreadsheets/d/1DqYgQnEDLQVQm5acMAhLgHLD8xXCG9BIrk-_Nv6jF3k/edit#gid=224728652 [Accessed: 2023-11-28]

[4]Amazon EC2 Spot Price History: https://zenodo.org/doi/10.5281/zenodo.5880792

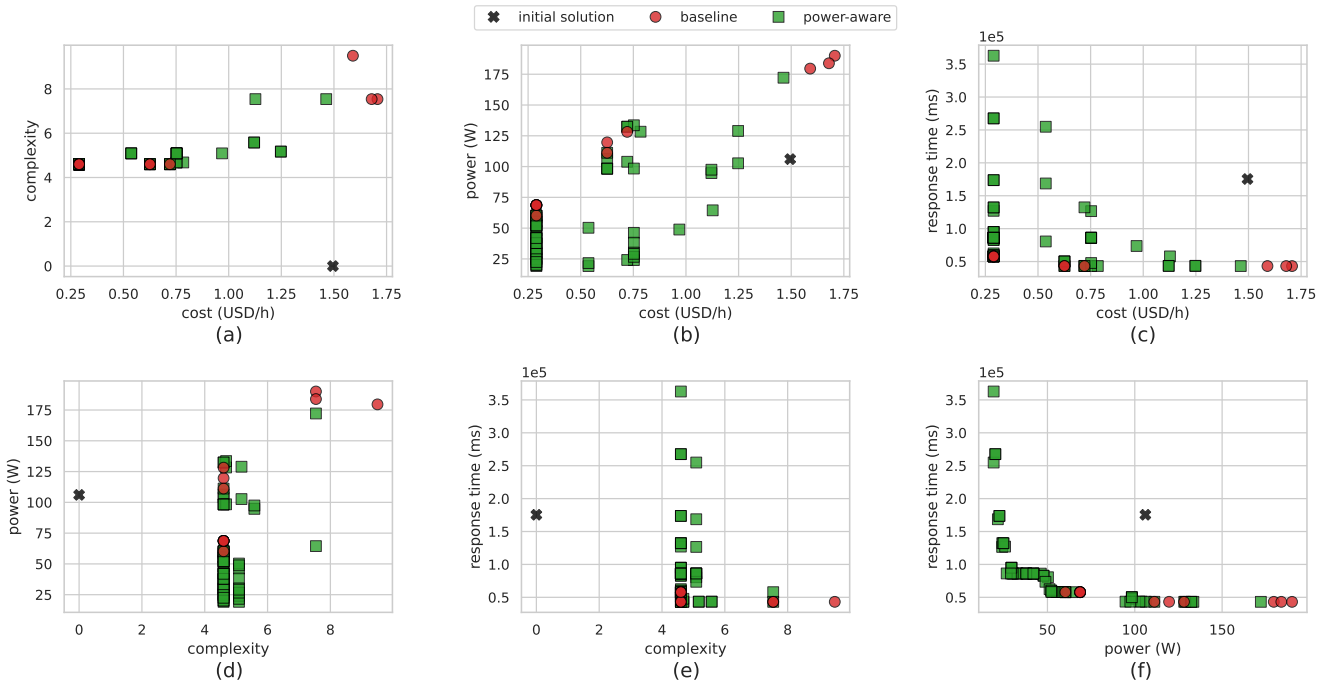


Fig. 2. Comparison of the Pareto fronts resulting from the *baseline* (without the power objective, 13 solutions) and the *power-aware* (with the power objective, 68 solutions) experiments. In the *baseline* experiment, power consumption was not considered as an optimization objective, but only computed afterward on the models that are part of the Pareto front.

A first noticeable difference between the Pareto fronts is that the *power-aware* experiment has a larger number of solutions (68) than the *baseline* experiment (13). This is expected in most cases, because solutions found with a larger number of objectives live in a higher dimensional space where it is more difficult to be dominated. A larger number of objectives is also likely to result in Pareto fronts which are more spread out in the space, as it is the case in our experiments.

The shapes of the Pareto fronts are quite noticeable for the *power-aware* experiment when *power* and *cost* are compared to *response time* (Figures 2c and 2f, respectively), but not so much for the *baseline* experiment. Indeed, the *power-aware* experiment provides plenty of solutions to choose, with low power consumption, low response time, and a wide range of costs. As expected, the *baseline* is able to find solutions with low response time, but half of the Pareto front contains solutions with higher cost (Figure 2c).

Somehow unexpectedly, instead, both the experiments show a correlation between cost and power consumption (Figure 2b). This is more evident in the *power-aware* experiment, and it is likely due to the trade-off in available cloud instances. Indeed, the current cloud offers, at a higher cost, instances with better performance and, most likely, higher power consumption.

Finally, the cost of changing the architecture (*complexity*) does not vary so much across the two experiments (Figure 2a and 2d). By focusing on *complexity*, Pareto fronts look more flat than in the other cases, most probably because this objective moves in discrete steps in the space, and not continuously as the other objectives.

B. Differences in the distributions of the objectives

Figure 3 shows the distributions of the values of the objectives in all the Pareto fronts of the 31 runs of the *baseline* and *power-aware* experiments. The distributions are shown as violin plots, with kernel density estimates, medians, and interquartile ranges as box plots inside violins.

While the *cost* objective is significantly more spread out in the *baseline*, the distributions of both experiments seem to be similar. The *complexity* objective exhibits a similar behavior in both experiments, with strikingly similar distributions. Power consumption, while having similar distribution shapes, it is shifted in location towards lower values in the *power-aware* experiment, as expected. Finally, the *response time* objective is the one that shows the most significant difference across the two experiments. In this case, the *power-aware* experiment has a higher median and a more spread out distribution, with a longer tail towards higher values, indicating a more diverse Pareto front with respect to this objective.

C. Quantifying the effect of considering power consumption as an additional objective

In order to provide a more precise estimation of the potential loss in performance and cost that may result from taking into account the additional objective of power consumption, we introduce the *prospective sustainability penalty (PSP)* metric. Given an initial architecture and an optimization objective,

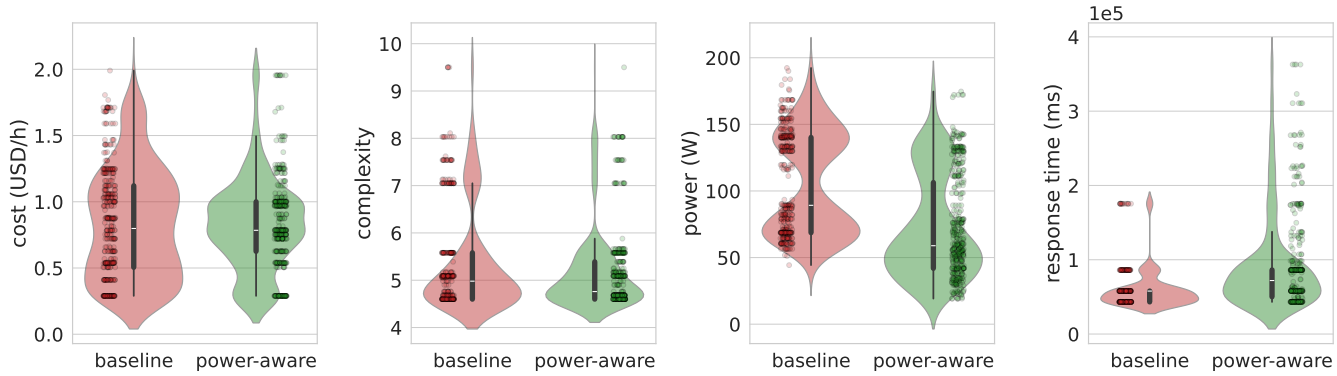


Fig. 3. Distributions of the objective values in the *baseline* and the *power-aware* experiments. Plotted solutions are all the Pareto fronts obtained in the 31 runs.

PSP is defined through two characteristics: difference and magnitude.

PSP difference: This is an estimate of the difference in the objective values when the power objective is added to the optimization problem. It is obtained by computing the Hodges-Lehmann estimator [32] of the difference between the values of the objective in the *baseline* and *power-aware* experiments. The Hodges-Lehmann estimator is a robust estimator of the median of the distribution of the differences between two samples. It is defined as the median of all the pairwise differences between the samples. It is robust to outliers and does not require the distributions of the samples to be normal. The formula for the Hodges-Lehmann estimator is:

$$\mathrm{HL}(X,Y) = \mathrm{median}\left(\{x_i - y_j \mid x_i \in X, y_j \in Y\}\right) \quad (1)$$

In our case, X and Y are the values of the objective in the Pareto fronts of the *baseline* and *power-aware* experiments, respectively.

PSP magnitude: This is an estimate of the magnitude of the differences in the objective values when the power objective is added to the optimization problem. This is obtained by first performing the Mann-Whitney U test [33] on the values of the objective obtained for the Pareto fronts of each run in the *baseline* and *power-aware* experiments, and then by computing a measure of effect size. The Mann-Whitney U test is a non-parametric test that can be used to determine whether two independent samples were drawn from a population with the same distribution. The null hypothesis of the test is that the two samples were drawn from the same distribution. The alternative hypothesis is that the two samples were drawn from distributions with different medians. The test returns a p-value, which is the probability of observing the given samples if the null hypothesis is true. As it is customary, we set the threshold of the p-value to 0.05. If the null hypothesis is rejected, thus meaning that there was a significant difference between the *baseline* and *power-aware* experiments, then the PSP magnitude is defined as the Cliff's delta [34]. The Cliff's delta is a non-parametric effect size measure that can be used to quantify the magnitude of the difference between two groups. It ranges from -1 to 1, with positive values indicating a tendency for the first group to have larger values, and negative values indicating a tendency for the second group to have larger values. The formula to compute the Cliff's delta (δ) from the Mann-Whitney U statistic (U) is:

$$\delta = \frac{2U}{n_1 n_2} - 1 \quad (2)$$

where n_1 and n_2 are the sizes of the two samples. If the null hypothesis is not rejected, the PSP magnitude is defined as 0. We interpret the Cliff's delta as follows: $\delta < 0.147$ is a negligible effect, $0.147 \leq \delta < 0.33$ is a small effect, $0.33 \leq \delta < 0.474$ is a medium effect, and $\delta \geq 0.474$ is a large effect [35].

Therefore, given an initial architecture A_0 and the additional optimization objective $power$, the PSP of the objective obj is defined as a pair:

$$\mathrm{PSP}_{power}(A_0, \mathrm{obj}) = [\mathrm{HL}(X,Y), \delta] \quad (3)$$

where X and Y are the values of the objective in the Pareto fronts of the experiment with and without $power$, respectively. Table II reports the PSP_{power} for each objective, along with the difference in the mean, and the other statistics that were used to compute the PSP. The *power* objective obviously has a large PSP, as expected. Other than that, the *response time* objective has a medium PSP, with a Cliff's delta of 0.46, which is very close to the threshold of 0.474 that separates medium and large effects.

Summary: In our experiments on the Train Ticket Booking Service case study, we observed that the addition of the power objective to the optimization problem has a significant impact on the response time of the system. However, contrary to what one might expect, the impact on the deployment cost is negligible. This leads to assume that, in order to obtain more sustainable solutions, we would have to trade in performance, but not in cost.

TABLE II
DESCRIPTIVE STATISTICS AND PROSPECTIVE SUSTAINABILITY PENALTY (PSP) OF THE OBJECTIVES IN THE *baseline* AND *power-aware* EXPERIMENTS. PSP IS COMPUTED AS THE HODGES-LEHMANN ESTIMATOR OF THE DIFFERENCE BETWEEN THE VALUES OF THE OBJECTIVE IN ALL THE PARETO FRONTS OF THE 31 RUNS OF THE *baseline* AND *power-aware* EXPERIMENTS, AND THE CLIFF'S DELTA.

objective	mean difference	HL	MWU p-value	Cliff's delta	PSP
cost (USD/h)	-0.185940	0.000000	0.737400	-0.013435	0.00, -0.01 (negligible)
complexity	-0.621584	0.000000	0.546571	0.023641	0.00, 0.02 (negligible)
power (W)	-49.568407	-30.318120	0.000000	-0.529768	-30.32, -0.53 (large)
response time (ms)	38611.479219	14684.824000	0.000000	0.459672	14684.82, 0.46 (medium)

V. WHAT IS THE EFFECT OF REFACTORING ACTIONS ON THE DISTRIBUTION OF POWER AND COST ACROSS DIFFERENT TYPES OF REQUESTS? (RQ2)

Individual functionalities contribute to the overall power consumption and cost of a system in different ways, because they are used with different frequencies and intensities, and they need different resources to be executed. Each functionality can be associated to a type of request (*i.e.*, a scenario or a Sequence Diagram in UML), and the power consumption and cost of a type of request can be computed by aggregating the utilization of the resources that are used to serve that type of request. In this section, we describe how we attribute power consumption and cost to individual types of requests. Such information can be used to study how these two properties relate to user behavior, and provide an additional view on the trade-offs that should be considered when optimizing the system.

A. Attributing power consumption and cost to types of requests

We associate power consumption and cost to individual types of requests by adapting the models used for the optimization in Section II-A. The amount of time that an instance is busy serving a type of request can be used to derive the share of the system power and cost that is spent throughout the system to serve that request. This information can be obtained by reconstructing the flow of requests in the LQN that are generated from our UML architectures. In the specification of our architecture, different types of requests are modelled through UML Sequence and Deployment Diagrams, and UML Nodes represent cloud instances. As schematized in Figure 4, a UML Node is translated into a *processor* in LQN, a UML Component into a *task*, and a message in a Sequence Diagram becomes an *entry* in the task. When the LQN model is solved to compute performance indices, the solver annotates such indices back on the model. Among the indices that the solver can compute, we are interested to the utilization of processors (U_p), and in particular to the share of this utilization associated to individual entries deployed on the processor ($U_{e,p}$). Given a processor p and an entry e, the contribution of an entry to the power consumption of the processor it is deployed on can be defined as:

$$\text{power}(e) = U_{e,p} * \text{power}_{max}(p) + (1 - U_p) \cdot k \cdot \text{power}_{max}(p)) \cdot U_{e,p}/U_p \quad (4)$$

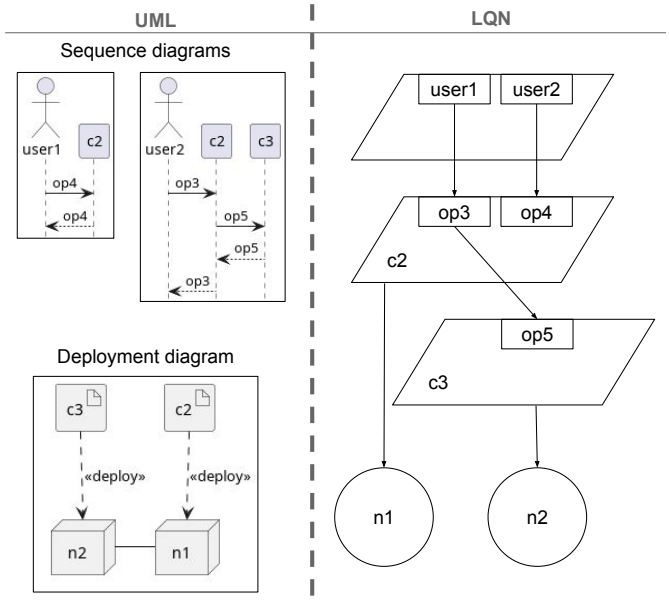


Fig. 4. Simplified view of the transformation from UML to LQN.

where $\text{power}_{max}(p)$ is the power consumption of the processor p when busy, and k is the factor by which scaling the power when the processor is idle. The idea is to use the utilization of an entry to compute its power consumption when busy, and then sharing the idle time of the processor proportionally among the entries that are using it [15]. The flow of a type of requests r can be defined as a sequence $S =< e_1, e_2, ..., e_n >$ of entries e, each deployed on a processor p. Accordingly, the power consumption of a type of requests r can be defined as:

$$\text{power}(r) = \sum_{i=1}^{|S|} \text{power}(e_i) \qquad \forall e_i \in S \quad (5)$$

that is the sum of the power consumption of all the entries invoked to satisfy the request r triggered by a given scenario. A similar, but simpler, reasoning can be applied to compute the cost of a type of requests. While the operating cost of the entire systems is computed as the sum of the cost of all the deployed nodes, the utilization of entries can be used again to attribute shares of that cost. Therefore, the cost of a type of

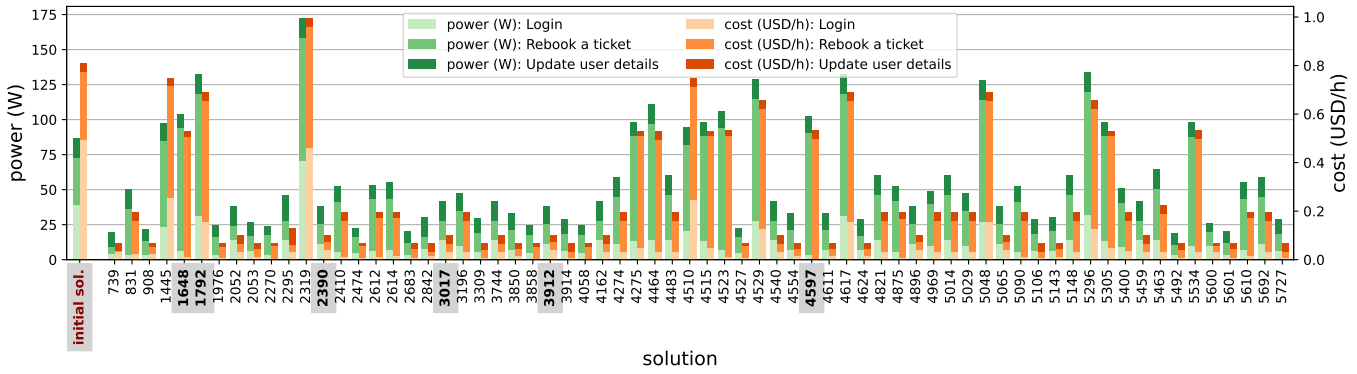


Fig. 5. Power consumption and cost of individual types of requests for the solutions in the Pareto front of the *power-aware* experiment. Entire bars represent the total power consumption and cost of the system for a given solution. The x-axis lists solutions IDs, highlighted in bold if mentioned in the text.

requests r can be defined as:

$$\text{cost}(r) = \sum_{i=1}^{|S|} \text{cost}(e_i) \qquad \forall e_i \in S \tag{6}$$

where $\text{cost}(e_i) = \text{cost}(p) \cdot U_{e,p}/U_p$.

B. User profiles

Table III shows the mean, standard deviation, and median of power consumption and cost of individual types of requests (*Scenarios*) in the super Pareto front of the *power-aware* experiment, along with the values for the initial solution. On average, both cost and power consumption are reduced by the optimization, but with a high variability across the different types of requests. The standard deviation of the *Rebook a ticket* scenario is particularly high, most probably because this scenario is more complex than the other ones, and therefore more sensitive to changes in the architecture. With such differences, it is important to let the designer aware of all the different trade-offs that can be made when optimizing the system, and that are not visible when looking at the overall power consumption and cost of the system.

TABLE III
SUMMARY OF POWER CONSUMPTION AND COST OF INDIVIDUAL TYPES OF REQUESTS FOR THE INITIAL SOLUTION AND FOR THE SOLUTIONS IN THE SUPER PARETO FRONT OF THE *power-aware* EXPERIMENT.

Scenario	Initial solution	*power-aware* experiment		
		mean	std	median
cost: Login	0.495	0.045	0.073	0.032
cost: Rebook a ticket	0.279	0.166	0.182	0.088
cost: Update user details	0.036	0.029	0.009	0.033
power: Login	38.791	11.610	10.049	9.727
power: Rebook a ticket	34.127	33.913	28.264	19.828
power: Update user details	13.815	11.441	2.508	11.883

Figure 5 shows a more detailed view of the same experiment. It is important to remind that the optimization algorithm does not alter the number of requests of each type or their frequency, but only the way they are served by applying the refactoring actions defined in Section II-C. Nonetheless, as it can be seen in the figure, the share of power consumption and cost of individual types of requests varies significantly across the solutions in the Pareto front. This is due to the fact that the optimization impacts on the utilization of the resources that are used to serve each type of request and, consequently, on their shares of power consumption and cost.

In some configurations, a single type of request appears to be responsible for most of the power consumption and cost of the entire system, as it is the case for the *Rebook a ticket* scenario in solutions with higher values of power and cost (*e.g.*, solutions 1648, 1792, 4597). Conversely, there are cases with lower values of power and cost that are characterized by more balanced shares among the requests (*e.g.*, solutions 2390, 3017, 3912).

These cases in which different types of requests end up contributing to the overall power consumption and cost of the system in opposite ways would be very hard to spot without the support of the optimization, whereas they are fundamental to study the trade-offs that can be made when optimizing the system. For example, this awareness would allow architectural designers to decide that the cost and power consumption of a certain type of request is too high with respect to its priority/relevance in the system and, as a consequence, a limitation on the concurrent number of these requests can be introduced.

One possible reason for those disruptive changes in terms of power consumption and cost might be pinpointed to refactoring actions that saturate the utilization of some nodes in favor of cost reductions. In other terms, the tendency of the optimization to minimize the number of nodes in order to reduce the cost of the system has the drawback of increasing their utilizations, thus possibly causing an increase in global power consumption.

Summary: We have shown how the power consumption and cost of individual types of requests can be obtained on the basis of the utilization of the resources that serve them. This information can be used to study how these two properties relate to user behavior, thus to provide an additional view on system trade-offs that are not visible when looking at the

overall power consumption and cost of the system. Such a view enables architectural decisions aimed at fine-tuning the requirement satisfaction of different types of users.

VI. How do the architectural solutions change when introducing power consumption among the optimization objectives? (RQ3)

In this research question, we take a look at how the optimization algorithm employs the refactoring catalog at its disposal (Section II-C) for sake of sustainability. Indeed, we intend to extract possible insights about refactoring actions that seem to be beneficial in reducing power consumption. Specifically, we focus on refactoring actions that are more frequently used by the algorithm and how they correlate with the objectives.

TABLE IV
Comparison of the frequencies of types of refactoring actions in the super Pareto fronts of the *baseline* and *power-aware* experiments. Target elements of the actions (*Target* column), and where those elements were relocated (*To* column) report the type of UML element as Node (N), Component (C), and Operation (O). Frequency column also shows the number of occurrences of the action type in the super Pareto front. Frequencies are reported for both experiments when the refactoring action and its target element are the same.

Type	Target (N,C,O)	To (N,C)	Frequency	
			baseline	power-aware
DROP	(N) verification	—	25.00% (13)	24.26% (66)
DROP	(N) login	—	21.15% (11)	21.69% (59)
DROP	(N) order-other	—	19.23% (10)	24.63% (67)
DROP	(N) route-plan	—	13.46% (7)	15.81% (43)
REDO	(C) order-other	(N) new-node	7.69% (4)	0.74% (2)
DROP	(N) travel-plan	—	3.85% (2)	2.57% (7)
MOVE	(O) login	(C) ticket-info	1.92% (1)	0.37% (1)
MOVE	(O) updateuser	(C) travel-plan	1.92% (1)	—
DROP	(N) rebook	—	1.92% (1)	—
DROP	(N) sso	—	1.92% (1)	—
DROP	(N) ticket-info	—	1.92% (1)	0.74% (2)
MOVE	(O) login	(C) verification	—	1.47% (4)
CLON	(N) login	—	—	1.47% (4)
MOVE	(O) getbyid	(C) rebook	—	1.47% (4)
MOVE	(O) login	(C) rebook	—	1.10% (3)
DROP	(N) seat	—	—	1.10% (3)
MOVE	(O) login	(C) travel-plan	—	0.74% (2)
MOVE	(O) rebook	(C) order-other	—	0.74% (2)
CLON	(N) ticket-info	—	—	0.37% (1)
MOVE	(O) login	(C) sso	—	0.37% (1)
MOVE	(O) modify	(C) station	—	0.37% (1)

Table IV shows the frequencies of the refactoring actions in the super Pareto fronts of the *baseline* and *power-aware* experiments. The removal of a node (*DROP*) is by far the most frequent action in both experiments. This is not surprising, as the removal of a node tends to reduce not only the power consumption of the system, but also its deployment cost.

When a node is removed, the approach has to relocate its components to other nodes, by preferring nodes hosting components that more frequently communicate with relocated ones. As explained in Section II-C, we have introduced such relocation criterion with the intent of not remotely spreading highly interacting components. The occurrence of so many removals of nodes might also indicate that the approach is able to identify nodes with a too low utilization to justify their cost. For instance, the component responsible for the CAPTCHA verification is initially hosted on the *verification* node. The latter is frequently removed and its component relocated to the *login* node, which is the only node that communicates with the *verification* node. The frequent occurrence of such refactoring action may indicate that the system is oversized and, eventually, the two components can be merged in one microservice.

When we compare the frequencies of the action that redeploys a component to another node (*REDO*), we can see that it is considerably more frequent in the *baseline* experiment, and that the component is relocated to a new node, thus adding a new node to the architecture. Conversely, when power consumption is considered, the approach tends to remove nodes instead of adding them, even if it may have a detrimental effect on the performance of the system.

The *MOTN* action, which moves an operation to a new component on a new node, does not appear in the super Pareto front of either experiment. This is probably due to the fact that the creation of a new node is a complex operation, and the approach tends to avoid it when possible.

Finally, we can see from the bottom half of Table IV that the *power-aware* experiment leads to a larger diversity in the number of refactoring actions, and to actions that are not present in the *baseline* experiment. From those actions that are specific to the Pareto front of *power-aware* experiment we can see, for instance, that the *login* component and node are often targets of different types of refactoring actions. This confirms that the login functionality has room for improvement with regard to power consumption.

Summary: We have observed that the refactoring actions that most frequently occur in the Pareto fronts are the ones that remove nodes from the architecture. This indicates that the algorithm is able to identify nodes that are hosting microservices that are too small to justify the cost and power consumption of a separate node. We have also seen that, when the power consumption is considered, different types of refactoring actions and target elements are used, which leads to discover architectures that were not appearing in the *baseline* experiment.

VII. Related Work

In the last decades, the problem of assessing sustainable systems is gaining more and more attention, as witnessed by the increasing number of studies in the literature [5]–[7], [36].

Chauhan *et al.* [7] highlighted challenges in designing systems, ranging from embedded systems to IoT devices. Additionally, the study pointed to untapped research potential in green computing, energy-efficient systems, mobile cloud computing, and the Internet of Things within the context of architecting cloud-based systems. Funke and Lago [36] addressed the crucial need to integrate sustainability aspects into architectural decision-making. Beyond technical expertise,

architecture knowledge requires practical experience in representing, communicating, and managing architectural decisions. Our approach exploits software architectures and gives to the designer a way to understanding the impact of sustainable constraints on software architectures.

Procaccianti *et al.* [37] codified a Green Architectural Tactics catalog, providing architects with a systematic framework to incorporate energy-efficient design principles and deploy reusable solutions for developing environmentally conscious software. Ponsard *et al.* [38] introduced a UML profile aimed at augmenting the UML with energy-aware concepts. We differentiate from the Procaccianti *et al.* study by introducing the automation concepts in applying refactoring actions on software architectures. Moreover, our refactoring actions could be mapped to some tactics in such catalog. Instead of using the Ponsard *et al.* UML profile, we exploited the MARTE profile by the OMG group [23], which is considered the standard profile for non-functional analysis with UML.

Houssein *et al.* [5] presented a comprehensive review, categorizing popular meta-heuristic techniques based on scheduling nature, objectives, task-resource mapping, and constraints. Task scheduling is crucial for optimal cloud service performance, yet improper scheduling can lead to resource underutilization or overutilization, resulting in wastage or degraded service. To address these challenges, meta-heuristic algorithms have been incorporated into task scheduling, efficiently distributing diverse tasks across limited resources. Zhan *et al.* [6] conducted a systematic mapping study to identify the state of the art of the optimization of virtualized resources within the cloud computing architecture. Moreover, Zhan *et al.* established a taxonomy at two levels for scheduling cloud resources and systematically reviews state-of-the-art approaches. Ram *et al.* [39] introduced the Investment-Based Optimization (IBO) meta-heuristic algorithm to optimize scheduling while minimizing execution costs while maximizing load across computing resources. Following these previous studies, we proposed a multi-objective approach to identify trade-offs between power consumption, response time, and cost while reducing refactoring actions complexity. Furthermore, our approach considers software architectures as first class of citizen.

Besides, we proposed a metric (see Section IV) to evaluate the impact that sustainability constraints could have on other aspects, such as performance and cost. The aim of such a metric is to provide to designers a quantifiable way to estimate improvement (detriment) that the sustainability has on their software systems.

VIII. Threats to validity

Construct validity: We acknowledge that the parameters utilized for enhancing the initial models with non-functional information could have introduced variability into our results, as all objectives are influenced by such information. To address this concern, we adopted model parametrization strategies from the literature [29], [30].

Another potential source of impact on our study is the genetic algorithm employed in our experiments. This algorithm plays a pivotal role in determining how solutions are generated by our approach, influencing the search space explored. To minimize this potential threat, we opted for NSGA-II, a widely utilized algorithm [40], [41] that proved to perform well in comparison with other multi-objective algorithms [42].

Additionally, we took precautionary measures aligned with established best practices in the literature [31], [43] to alleviate the algorithm's influence on the final results.

Internal validity: Our exploratory study was designed as a paired comparison between an optimization experiment that does not consider power consumption and one that does. A threat to the internal validity of our study is the possibility that the results obtained in RQ1 are not due to the introduction of power consumption as an optimization objective, but rather to the introduction of a new objective in general [44]. To mitigate this threat, in RQ3 we contextualized the refactoring actions that were selected by the optimization algorithm, and we found that the actions selected in the *power-aware* experiment are more aligned with the goal of reducing power consumption than the actions selected in the *baseline* experiment.

External validity: We only examined the results of the application of our approach to a single case study, albeit a large and widely used one. Our intention was to provide a proof of concept of the applicability of our approach to a real-world system, and showcase how it can be used to support decision-making in the context of sustainable deployment planning. However, we acknowledge that the results obtained may not be generalizable to other systems, and we plan further studies to validate the approach in other contexts.

Although our catalog of refactoring actions is relatively small, comprising only five actions, it is important to note that the complexity of potential modifications to the initial architectural model is not solely determined by the number of actions. The challenge lies in the vast solution spaces created by the multitude of possible sequences in which these refactoring actions can be applied. While a larger set of refactoring actions could theoretically capture more intricate changes, the associated increase in solution space would demand substantial computational resources, making it a trade-off between comprehensiveness and practical feasibility [45].

Conclusion validity: To mitigate the risk of drawing incorrect conclusions from our results, we adopted proper significance tests and effect size measures. Specifically, we employed the Hodges-Lehmann estimator [32] and the Cliff's delta effect size [34] to avoid making assumptions about the distribution of the data. We also reported p-values and effect sizes for all comparisons, and we only considered results with a confidence level of 95%.

IX. Conclusion

This paper introduces a novel approach that leverages NSGA-II to generate diversified deployment configurations of a software architecture through refactoring actions, while aiming to provide optimal trade-offs between system performance, deployment costs, complexity and power consumption. The exploration of the intricate relationships between power

consumption, cost, and user behavior is also supported, by looking at different types of user requests.

Results from our experiments on Train Ticket Booking Service application indicate that sustainability objectives, particularly focusing on power consumption, significantly impact system response time. Surprisingly, this impact is observed with negligible effects on deployment costs.

As future work, we plan to evaluate the proposed approach on other case studies, and to investigate the impact of the proposed approach on the evolution of a software architecture over time. Moreover, the awareness of the impact of deployment decisions on the cost and power consumption of serving certain types of requests opens to the possibility of using this information in a more fine-grained refactoring process that aims at optimizing towards specific user interaction scenarios. Finally, by highlighting the variation in the contribution of individual types of requests, our approach could also identify opportunities for merging smaller microservices to save on power consumption and costs.

ACKNOWLEDGMENTS

This work has been funded by European Union – NextGenerationEU – National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza, PNRR) – Project: "SoBigData.it – Strengthening the Italian RI for Social Mining and Big Data Analytics" – Prot. IR0000013 – Avviso n. 3264 del 28/12/2021, and Italian Government (Ministero dell'Università e della Ricerca, PRIN 2022 PNRR) – cod.P2022SELA7: "RECHARGE: monitoRing, tEsting, and CHaracterization of performAnce Regressions" – Decreto Direttoriale n. 1205 del 28/7/2023.

REFERENCES

[1] L. Belkhir and A. Elmeligi, "Assessing ict global emissions footprint: Trends to 2040 & recommendations," *Journal of Cleaner Production*, vol. 177, pp. 448–463, 2018, ISSN: 0959-6526. DOI: https://doi.org/10.1016/j.jclepro.2017.12.239.

[2] D. Rodrìguez-Gracia, J. Piedra-Fernandez, L. Iribarne, J. Criado, R. Ayala, J. Alonso-Montesinos, and C.-U. M. de las Mercedes, "Microservices and machine learning algorithms for adaptive green buildings," *Sustainability*, 2019. DOI: 10.3390/SU11164320.

[3] X. Zhou, X. Peng, T. Xie, J. Sun, C. Xu, C. Ji, and W. Zhao, "Benchmarking microservice systems for software engineering research," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ser. ICSE '18, Gothenburg, Sweden, 2018, pp. 323–324, ISBN: 9781450356633. DOI: 10.1145/3183440.3194991.

[4] D. Mytton, "Assessing the suitability of the greenhouse gas protocol for calculation of emissions from public cloud computing workloads," *Journal of Cloud Computing*, vol. 9, pp. 1–11, 2020. DOI: 10.1186/s13677-020-00185-8.

[5] E. H. Houssein, A. G. Gad, Y. M. Wazery, and P. N. Suganthan, "Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends," *Swarm and Evolutionary Computation*, vol. 62, p. 100 841, 2021, ISSN: 2210-6502. DOI: https://doi.org/10.1016/j.swevo.2021.100841.

[6] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Comput. Surv.*, vol. 47, no. 4, 2015, ISSN: 0360-0300. DOI: 10.1145/2788397.

[7] M. A. Chauhan, M. A. Babar, and B. Benatallah, "Architecting cloud-enabled systems: A systematic survey of challenges and solutions," *Software: Practice and Experience*, vol. 47, no. 4, pp. 599–644, 2017. DOI: https://doi.org/10.1002/spe.2409. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2409.

[8] G. Huppes and M. Ishikawa, "A framework for quantified eco-efficiency analysis," *Journal of Industrial Ecology*, vol. 9, 2005. DOI: 10.1162/108819805775247882.

[9] X. Xu, W. Dou, X. Zhang, and J. Chen, "Enreal: An energy-aware resource allocation method for scientific workflow executions in cloud environment," *IEEE Transactions on Cloud Computing*, vol. 4, pp. 166–179, 2016. DOI: 10.1109/TCC.2015.2453966.

[10] J. Baliga, R. Ayre, K. Hinton, and R. Tucker, "Green cloud computing: Balancing energy in processing, storage, and transport," *Proceedings of the IEEE*, vol. 99, pp. 149–167, 2011. DOI: 10.1109/JPROC.2010.2060451.

[11] D. Armstrong, K. Djemame, and R. Kavanagh, "Towards energy aware cloud computing application construction," *Journal of Cloud Computing*, vol. 6, pp. 1–13, 2017. DOI: 10.1186/s13677-017-0083-2.

[12] M. Hähnel, F. M. Arega, W. Dargie, R. Khasanov, and J. Castrillón, "Application interference analysis: Towards energy-efficient workload management on heterogeneous micro-server architectures," *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 432–437, 2017. DOI: 10.1109/INFCOMW.2017.8116415.

[13] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: NSGA-II," in *Parallel Problem Solving from Nature - PPSN VI, 6th International Conference, Paris, France, September 18-20, 2000, Proceedings*, ser. Lecture Notes in Computer Science, vol. 1917, 2000, pp. 849–858. DOI: 10.1007/3-540-45356-3_83.

[14] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: Past, present, and future," *Multim. Tools Appl.*, vol. 80, no. 5, pp. 8091–8126, 2021. DOI: 10.1007/s11042-020-10139-6.

[15] H. Xu and C. Jian, "A meta reinforcement learning-based virtual machine placement algorithm in mobile edge computing," *Cluster Computing*, 2023, ISSN: 1573-7543. DOI: 10.1007/s10586-023-04030-w.

[16] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012, Special Section: Energy efficiency in large-scale distributed systems, ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2011.04.017.

[17] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ser. ISCA '07, San Diego, California, USA, 2007, pp. 13–23, ISBN: 9781595937063. DOI: 10.1145/1250662.1250665.

[18] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," vol. 35, no. 2, pp. 13–23, 2007, ISSN: 0163-5964. DOI: 10.1145/1273440.1250665.

[19] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece, *Software cost estimation with COCOMO II*. 2009.

[20] A. Trendowicz, *Software Cost Estimation, Benchmarking, and Risk Assessment: The Software Decision-Makers' Guide to Predictable Software Development*. 2013.

[21] D. Di Pompeo and M. Tucci, "Search budget in multi-objective refactoring optimization: A model-based empirical study," in *48th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2022, Maspalomas, Gran Canaria, Spain, 31 August - 2 September 2022*, 2022, pp. 406–413. DOI: 10.1109/SEAA56994.2022.00070.

[22] A. Rago, S. A. Vidal, J. A. Diaz-Pace, S. Frank, and A. van Hoorn, "Distributed quality-attribute optimization of software architectures," in *Proceedings of the 11th Brazilian Symposium on Software Components, Architectures and Reuse, SBCARS 2017, Fortaleza, CE, Brazil, September 18 - 19, 2017*, 2017, 7:1–7:10. DOI: 10.1145/3132498.3132509.

[23] O. M. Group, *A UML profile for MARTE: modeling and analysis of real-time embedded systems*, Object Management Group, 2008.

[24] G. Franks and C. M. Woodside, "Multiclass multiservers with deferred operations in layered queueing networks, with software system appli-

cations," in *12th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2004), 4-8 October 2004, Vollendam, The Netherlands*, 2004, pp. 239–248. DOI: 10.1109/MASCOT.2004.1348262.

[25] C. Li, T. Altamimi, M. H. Zargari, G. Casale, and D. C. Petriu, "Tulsa: A tool for transforming UML to layered queueing networks for performance analysis of data intensive applications," in *Quantitative Evaluation of Systems - 14th International Conference, QEST 2017, Berlin, Germany, September 5-7, 2017, Proceedings*, ser. Lecture Notes in Computer Science, vol. 10503, 2017, pp. 295–299. DOI: 10.1007/978-3-319-66335-7_18.

[26] M. Mazkatli and A. Koziolek, "Continuous integration of performance model," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '18, Berlin, Germany: Association for Computing Machinery, 2018, pp. 153–158, ISBN: 9781450356299. DOI: 10.1145/3185768.3186285. [Online]. Available: https://doi.org/10.1145/3185768.3186285.

[27] L. Bortolussi, V. Galpin, J. Hillston, and M. Tribastone, "Hybrid semantics for PEPA," in *QEST 2010, Seventh International Conference on the Quantitative Evaluation of Systems, Williamsburg, Virginia, USA, 15-18 September 2010*, IEEE Computer Society, 2010, pp. 181–190. DOI: 10.1109/QEST.2010.31. [Online]. Available: https://doi.org/10.1109/QEST.2010.31.

[28] V. Cortellessa, D. Di Pompeo, V. Stoico, and M. Tucci, "Many-objective optimization of non-functional attributes based on refactoring of software models," *Inf. Softw. Technol.*, vol. 157, p. 107 159, 2023. DOI: 10.1016/j.infsof.2023.107159.

[29] F. Willnecker, M. Dlugi, A. Brunnert, S. Spinner, S. Kounev, W. Gottesheim, and H. Krcmar, "Comparing the accuracy of resource demand measurement and estimation techniques," in *Computer Performance Engineering - 12th European Workshop, EPEW 2015, Madrid, Spain, August 31 - September 1, 2015, Proceedings*, ser. Lecture Notes in Computer Science, vol. 9272, 2015, pp. 115–129. DOI: 10.1007/978-3-319-23267-6_8.

[30] V. Cortellessa, D. Di Pompeo, R. Eramo, and M. Tucci, "A model-driven approach for continuous performance engineering in microservice-based systems," *J. Syst. Softw.*, vol. 183, p. 111 084, 2022. DOI: 10.1016/J.JSS.2021.111084.

[31] A. Arcuri and G. Fraser, "On parameter tuning in search based software engineering," in *Search Based Software Engineering - Third International Symposium, SSBSE 2011, Szeged, Hungary, September 10-12, 2011. Proceedings*, ser. Lecture Notes in Computer Science, vol. 6956, 2011, pp. 33–47. DOI: 10.1007/978-3-642-23716-4_6.

[32] J. L. Hodges Jr and E. L. Lehmann, "Estimates of location based on rank tests," in *Selected Works of EL Lehmann*, 2011, pp. 287–300.

[33] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[34] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions.," *Psychological bulletin*, vol. 114, no. 3, p. 494, 1993.

[35] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys," in *annual meeting of the Florida Association of Institutional Research*, vol. 177, 2006, p. 34.

[36] M. Funke and P. Lago, "Carving sustainability into architecture knowledge practice," in *Software Architecture - 17th European Conference, ECSA 2023, Istanbul, Turkey, September 18-22, 2023, Proceedings*, ser. Lecture Notes in Computer Science, vol. 14212, 2023, pp. 54–69. DOI: 10.1007/978-3-031-42592-9_4.

[37] G. Procaccianti, P. Lago, and G. A. Lewis, "A catalogue of green architectural tactics for the cloud," in *2014 IEEE 8th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems*, 2014, pp. 29–36. DOI: 10.1109/MESOCA.2014.12.

[38] C. Ponsard, R. De Landtsheer, G. Ospina, and J.-C. Deprez, "Towards design-time simulation support for energy-aware cloud application development," vol. 2, 2016, pp. 398–404. DOI: 10.5220/0005933503980404.

[39] S. D. K. Ram, S. Srivastava, and K. K. Mishra, "A new meta-heuristic approach for load aware-cost effective workflow scheduling," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 21, e7112, 2022. DOI: https://doi.org/10.1002/cpe.7112. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.7112.

[40] A. Koziolek, H. Koziolek, and R. H. Reussner, "Peropteryx: Automated application of tactics in multi-objective software architecture optimization," in *7th International Conference on the Quality of Software Architectures, QoSA 2011 and 2nd International Symposium on Architecting Critical Systems, ISARCS 2011. Boulder, CO, USA, June 20-24, 2011, Proceedings*, 2011, pp. 33–42. DOI: 10.1145/2000259.2000267.

[41] A. Ouni, M. Kessentini, M. Ó. Cinnéide, H. A. Sahraoui, K. Deb, and K. Inoue, "MORE: A multi-objective refactoring recommendation approach to introducing design patterns and fixing code smells," *J. Softw. Evol. Process.*, vol. 29, no. 5, 2017. DOI: 10.1002/SMR.1843.

[42] T. Hiroyasu, S. Nakayama, and M. Miki, "Comparison study of spea2+, spea2, and NSGA-II in diesel engine emissions and fuel economy problem," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005, 2-4 September 2005, Edinburgh, UK*, 2005, pp. 236–242. DOI: 10.1109/CEC.2005.1554690.

[43] A. Arcuri and L. C. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, 2011, pp. 1–10. DOI: 10.1145/1985793.1985795.

[44] Y.-J. Lai, T.-Y. Liu, and C. Hwang, "Topsis for modm," *European Journal of Operational Research*, vol. 76, pp. 486–500, 1994. DOI: 10.1016/0377-2217(94)90282-8.

[45] L. Marti, J. García, A. Berlanga, and J. M. Molina, "An approach to stopping criteria for multi-objective optimization evolutionary algorithms: The MGBM criterion," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2009, Trondheim, Norway, 18-21 May, 2009*, 2009, pp. 1263–1270. DOI: 10.1109/CEC.2009.4983090.