Energy-Aware Application Life-Cycle Management in Cloud Edge Computing Continuum Using Applications Profiles

Mohamed Mekki, *Member, IEEE,* Adlen Ksentini, *Senior Member, IEEE* Miloud Bagaa, *Senior Member, IEEE*

Abstract—Efficient management of an application's life-cycle on a Cloud Edge Computing Continuum (CECC) infrastructure requires knowledge about: (i) the application behavior as well as its requirements in terms of compute and network resources evolution over time; (ii) information about the state and characteristics of the infrastructure, including its produced carbon footprint. In this paper, we propose an architecture for the CECC Application Orchestrator to efficiently handle the application's life cycle by designing and relying on applications and infrastructure profiling, which is done using monitoring information, including energy metrics and carbon intensity of infrastructures. First, we define a modeling method for application profiles, which represent the application's current and future compute and network requirements. Second, we model the problem of selecting the best locations to deploy or migrate applications while minimizing the deployment's cost and the carbon footprint. The model decides the current and future placements of each application and its allocated resources based on the application profile, the availability constraints of the application, and the available infrastructure resources. Last, we propose a heuristic solution to solve the problem rapidly, and we compare the different trade-offs between carbon and cost efficiency.

Index Terms—Carbon Footprint, Energy, Application Profiling, Cloud Edge Computing Continuum, Orchestration and Multi-objective optimization.

1 Introduction

Cloud Edge Computing Continuum (CECC) has seen light thanks to the paradigm shift in application development and deployment, integrating the scalability and resource availability of cloud computing with the real-time processing capabilities of edge computing [1]. This continuum is made possible due to the maturation of cloud and edge computing provisioning and management [2], providing a spectrum wherein workloads can seamlessly migrate between centralized cloud infrastructures and decentralized edge and far edge infrastructures or devices based on factors such as compute resources requirements, latency, and bandwidth constraints [3]. At one end of this continuum lies traditional cloud computing, characterized by massive data centers and centralized processing resources, offering resources scalability and storage capabilities. In contrast, on the other hand, edge computing leverages local computing infrastructures composed of small data centers or far-edge devices, reducing latency and enhancing responsiveness for time-sensitive applications. Additionally, edge and far-edge devices may be mobile or volatile, meaning that they are not available permanently due to either battery restrictions or mobility, such as for drone-based far-edge devices or single-board computers powered by batteries [4].

The continuum between these extremes enables new architecture that takes advantage of the dynamic distribution of computing tasks across cloud and edge nodes based on workload demands and network conditions. This paradigm encourages the development of innovative applications that span multiple domains such as IoT, autonomous systems, and augmented reality. In this case, seamless orchestration of resources across the cloud edge continuum is crucial for meeting performance requirements and enabling pervasive connectivity for deployed applications. The orchestration is handled by a CECC Orchestrator, responsible for managing the life cycle of applications deployed on the CECC infrastructure [5]. In order to efficiently manage the applications, the CECC or-

- *Mohamed Mekki is with Communication System Department, EURECOM, Sophia Antipolis, France. E-mail: mohamed.mekki@eurecom.fr*
- *Adlen Ksentini is with Communication System Department, EURECOM, Sophia Antipolis, France. E-mail: adlen.ksentini@eurecom.fr*
- *Miloud Bagaa is with Electrical and Computer Engineering Department, Université du Québec à Trois-Rivières, Quebec, Canada. E-mail: miloud.bagaa@uqtr.ca*

manuscript received: ;

chestrator should have information about the application profiles. Application profiling involves systematic analysis and characterization of applications deployed in cloud computing environments. This process aims to gain a deep understanding of an application's behavior, performance, resource usage, and dependencies within the cloud infrastructure [6] [7]. By profiling cloud applications, the CECC orchestrator can optimize the placement of the applications to improve their performance and optimize the infrastructure's resource usage. Profiling techniques may include monitoring system metrics and analyzing network traffic patterns [8]. While AI techniques can be used in order to predict the future needs of the application in terms of computing resources and network requirements. The Cloud Edge Computing Continuum infrastructure spans multiple locations and domains, including public cloud, dedicated edge cloud, and computing-capable devices. Each compute location is composed of nodes which are machines that consume electricity and water to function as well as for cooling. Based on the sources of energy used by the infrastructure modes, the CECC infrastructure and the workloads running on top of it produce a carbon footprint. In fact, recently, more and more governments are taking action to reduce carbon emissions; for example, the EU aims to be climate neutral by 2050; this objective is at the heart of The European Green Deal [9]. Communication technology is no exception, as in 2021, an ACM technology brief [10] estimated that the information and communication technology (ICT) sector contributed between 1.8% and 3.9% of global carbon emissions. Consequently, we believe that the CECC orchestrator should aim to minimize the carbon footprint of managed applications. To this end, energy monitoring systems such as Kepler [11] and infrastructure carbon intensity exporters can be used. The major contributions of the proposed work are the following:

1) We propose an architecture of CECC Application Orchestrator. The architecture leverages applications and infrastructure profiling to efficiently manage CECC applications. This profiling is performed using the monitoring information, including energy metrics and carbon intensity of the infrastructures.
2) We propose a modeling for application profiles from the point of view of the CECC Orchestrator; the profile represents the application's current and future compute and network requirements. The profile is constructed based on the historical use of compute and network resources by the application relying on statistical methods.
3) We profile infrastructure energy consumption and carbon footprint using the energy monitoring system Kepler [11] and the energy sources' carbon intensity.
4) We model the problem of selecting the best locations to deploy or migrate applications to while minimizing the deployment's cost and the carbon footprint. The model decides the current and future placements of each application based on the application profile, the availability constraints of the application, and the available infrastructure resources. Further, we propose a heuristic solution to solve the problem rapidly. We compare the different trade-offs between carbon and cost efficiency for different scenarios.

The rest of this article is structured as follows. Section 2 cites related works. Section 3 introduces the applications profiling model proposed in this work. Section 4 presents the architecture of the CECC Applications Orchestrator. The problem formulation with the objective functions is presented in section 5, it also includes the proposed heuristic to solve the problem. Section 6 illustrates the performance evaluation of the heuristic compared to the optimal solution. Finally, we conclude in section 7.

2 Related works

In this section, we review research addressing the reduction of carbon footprint and energy consumption across the cloud–edge/fog continuum. We structure the discussion along four main axes: (i) carbon and energy monitoring, (ii) carbon-aware and energy-aware application scheduling, (iii) serverless and Function-as-a-Service (FaaS) workloads, and (iv) energy-aware resource management, offloading, and service placement in fog and cloud–edge environments. We then discuss orchestration platforms for constrained and far-edge settings and conclude with a synthesis of gaps and trends. Table 1 summarizes the main dimensions captured by these works and contrast them with our proposal.

2.1 Carbon and Energy Monitoring in Cloud and Telco Environments

Early efforts focus on improving observability of energy consumption and carbon emissions at the infrastructure level. Kepler (Kubernetes-based Efficient Power Level Exporter) [11] is a node-level monitoring tool for Kubernetes clusters that collects real-time power metrics and exposes them as metrics for higher-level schedulers and controllers. By offering fine-grained energy visibility for containers and nodes, Kepler enables more informed, energy-aware workload management and forms a practical foundation for carbon-aware orchestration in cloud-native environments.

TABLE 1
Comparison of Related Works With Our Work

Reference	Support for Application Mobility	Energy-Aware Scheduling	Multi-Tier Deployment	Workload Profiling	Dynamic Adaptation	Serverless Workloads
[12]	✓	✓	✓	✗	✗	✗
[13]	✗	✓	✓	✗	✗	✗
[14]	✗	✓	✓	✗	✗	✗
[15]	✓	✓	✗	✗	✗	✗
[16]	✗	✓	✗	✗	✗	✗
[17]	✓	✓	✗	✗	✓	✓
[18]	✗	✗	✓	✗	✓	✓
[19]	✗	✓	✓	✗	✗	✗
[20]	✗	✓	✓	✓	✓	✗
[21]	✓	✓	✓	✗	✓	✗
[22]	✗	✓	✓	✓	✓	✗
[23]	✓	✓	✓	✗	✓	✗
[24]	✗	✓	✓	✗	✗	✗
[25]	✗	✓	✓	✓	✓	✗
[26]	✓	✓	✓	✓	✓	✗
[27]	✓	✓	✓	✓	✓	✗
[28]	✗	✓	✓	✓	✓	✗
[29]	✓	✓	✓	✗	✓	✗
[30]	✗	✓	✓	✓	✗	✗
[31]	✓	✗	✓	✗	✓	✓
[32]	✓	✗	✓	✗	✓	✓
[33]	✓	✓	✓	✓	✓	✗
[34]	✓	✓	✗	✗	✓	✗
[35]	✓	✓	✗	✓	✓	✗
[36]	✓	✓	✓	✓	✓	✗
[37]	✗	✓	✓	✗	✗	✗
[38]	✗	✓	✓	✗	✗	✗
[39]	✗	✓	✓	✗	✗	✗
[40]	✗	✓	✓	✗	✗	✗
[41]	✗	✓	✓	✗	✗	✗
[42]	✗	✓	✓	✗	✗	✗
[43]	✗	✓	✗	✗	✗	✗
[44]	✗	✓	✓	✗	✗	✗
[45]	✗	✓	✗	✗	✗	✗
[46]	✗	✓	✗	✗	✗	✗
[47]	✗	✓	✓	✗	✗	✗
[48]	✓	✓	✓	✓	✓	✗
[49]	✗	✓	✗	✗	✗	✗
[50]	✗	✓	✗	✗	✗	✗
our work	✓	✓	✓	✓	✓	✗

In the Telco domain, the work in [51] proposes a carbon emission monitoring framework for Telco Cloud infrastructures. It integrates power measurements of physical and virtual resources (VMs, VNFs, containers) with carbon intensity data from energy providers and uses machine-learning models to estimate and attribute carbon emissions to individual virtualized resources. Deployed in a production Internet Data Center, the system demonstrates accurate, real-time tracking of the environmental impact of Telco clouds and supports operators working toward net-zero objectives.

These monitoring solutions primarily address the *measurement* side of the problem, enabling later scheduling and placement policies to reason about power and carbon footprints.

2.2 Carbon-Aware Application and Workload Scheduling

A second line of work targets carbon-aware and energy-aware scheduling at the level of applications and workloads. KEIDS [12] extends the Kubernetes scheduler to multi-cluster edge cloud environments for Industrial IoT (IIoT). KEIDS is a carbon-aware, performance-conscious scheduler that considers energy efficiency and carbon footprint as first-class objectives when placing microservices across edge clusters. While it operates within Kubernetes and can be applied in Cloud–Edge Computing Continuum (CECC) infrastructures, it does not explicitly manage the full application lifecycle (e.g., long-term profiling, migrations, and autoscaling policies).

Several works exploit temporal variations in carbon intensity to shift workloads to greener periods or locations. The framework in [13] dynamically schedules cloud workloads to time windows with lower grid carbon intensity across multiple countries, focusing on batch and deferrable jobs. Similarly, the approach in [14] combines carbon footprint estimation with optimization and machine-learned predictions of carbon intensity to drive VM and container placement decisions. Both works achieve significant emission reductions by shifting non-urgent workloads to greener periods, but are not suitable for latency-sensitive or interactive applications that cannot be delayed.

At the edge, the decentralized peer offloading framework in [15] enables carbon-aware offloading across multiple providers without a centralized controller, preserving provider data privacy. However, it only considers a subset of continuum resources and does not explicitly model end-to-end multi-tier deployments. The work in [16] formulates a MILP model that jointly optimizes task offloading and energy sharing, including battery management, to minimize the carbon footprint of edge computing, using real carbon intensity traces from several European countries.

More recently, GreenScale [33] introduces a carbon-aware design space exploration framework for

edge–cloud applications. It models carbon emissions using time- and location-dependent carbon intensity, embodied emissions, workload characteristics, and runtime variance, and shows that carbon-optimal schedules can differ from energy- or performance-optimal ones. GreenScale highlights carbon efficiency as a distinct optimization axis but does not yet address application mobility, microservice lifecycle management, or full CECC orchestration.

2.3 Serverless and FaaS-Oriented Energy and Carbon Management

For Function-as-a-Service (FaaS) workloads, several solutions address energy or carbon efficiency in serverless environments. The framework in [17] targets energy-aware scheduling for serverless workloads at the edge. It considers device battery levels and renewable energy variability to guide placement of functions in Kubernetes and OpenFaaS clusters, demonstrating improved energy efficiency for edge serverless platforms.

CASA [18] proposes a dual-objective serverless framework that jointly optimizes Service Level Objectives (SLOs) and carbon emissions in cloud environments. CASA dynamically autoscales and schedules containers, switching between prewarming and cold starts using heuristic rules to minimize resource usage and associated emissions while controlling latency. It explicitly addresses the trade-off between performance and carbon footprint for serverless applications, but focuses mainly on cloud data centers and does not extend to heterogeneous edge or far-edge environments.

At a broader level, Oliveira et al. [31] conduct a systematic mapping study of FaaS platforms across the cloud-to-thing continuum. They show that existing FaaS solutions predominantly target cloud and edge layers, with limited support for resource-constrained IoT devices, and that placement, scaling, and orchestration mechanisms remain relatively static and infrastructure-centric. Filinis et al. [32] propose intent-driven orchestration of serverless applications in the computing continuum, using reinforcement learning-based controllers and multi-objective scheduling to map high-level intents to autoscaling and placement decisions. While these works advance continuum-aware serverless orchestration, they primarily optimize QoS, cost, and power, with limited emphasis on carbon-awareness and without comprehensive microservice-level lifecycle management.

2.4 Energy-Aware Resource Management and Offloading in Fog and Edge Systems

A large body of work focuses on energy-aware task and service management in fog and edge environments, often with latency and deadline constraints. Vadde et al. [19], Naha et al. [20], Bozorgchenani et al. [21], Gai et al. [22], Muhamad et al. [23], and Alshahrani et al. [24] explore multi-layer fog architectures and energy-aware offloading strategies. They leverage models and heuristics (e.g., multiple linear regression prediction [20], centralized vs. distributed offloading [21], time-constrained energy minimization [22], stable matching for 5G [23], and layered IoT–Fog–Cloud analysis [24]) to jointly optimize delay, energy consumption, and sometimes cost.

Other works refine task-level scheduling and resource allocation at the fog layer. Azizi et al. [25] propose deadline-aware semi-greedy schedulers (PSG and PSG-M) that reduce energy consumption and deadline violations for time-sensitive IoT workloads. Masri et al. [28] optimize communication energy by adapting the TCP Maximum Transmission Unit (MTU) in IoT–fog networks. Ullah et al. [27] leverage machine learning to predict service demand and dynamically switch fog nodes on or off to save energy, while Alsemmeari et al. [29] design heuristic service allocation mechanisms to favor fog-local processing and reduce backhaul traffic. Roumeliotis et al. [30] present a hybrid heuristic and multi-objective evolutionary framework for energy-efficient, deadline-aware task scheduling in fog systems.

These works demonstrate the effectiveness of predictive, heuristic, and optimization-based energy-aware resource management in fog environments. However, most operate on monolithic tasks or coarse services rather than microservices, and rarely integrate workload profiling, mobility, or carbon intensity into the decision process.

2.5 Energy-Aware Service Placement and Orchestration in the Cloud–Edge Continuum

Complementary to fog-centric scheduling, a growing set of works targets energy- and carbon-aware service placement and orchestration across the cloud–edge continuum. DECA [34] introduces a multi-objective placement method for edge clouds that jointly minimizes energy cost and carbon emissions, using A*-based search on application graphs. Self-optimising decentralized placement in cloud federations [35] and dynamic service migration in edge cognitive computing [36] rely on decentralized and adaptive migrations to improve performance and resource utilization, indirectly impacting energy. DPSO-based IoT service placement [37], MInRE [38], and discrete PSO (Particle Swarm Optimization) for MEC server placement [39] minimize compute energy while respecting delay or coverage constraints.

Further, Liu et al. [40] propose multi-objective offloading in fog environments; Mortazavi et al. [41] consider reliability-aware energy-efficient deployment; Natesha and Guddeti [42] use elitism-based GA

(Genetic Algorithm) for multiobjective IoT placement; and Pasteris et al. [43] provide placement algorithms with provable guarantees. Reddy et al. [44] optimize fog resource management; Saboor et al. [45] use rank-based microservice distribution to exploit a Green Energy Index; Shokouhifar [46] applies FH-ACO (Fuzzy Heuristic – Ant Colony Optimization) for joint VNF placement and routing; Taleb et al. [47] propose energy- and resource-aware graph-based microservice placement in the continuum; Tang et al. [48] design learning-based container migration; Alvarez Valera et al. [49] present DRACeo, a simulator for energy-saving microservice networks; and Yu et al. [50] jointly optimize routing and instance placement for microservices.

Collectively, these works demonstrate a rich design space of energy-aware and, in a few cases, carbon-aware placement and migration strategies. However, they often rely on static workloads and simplified energy models, rarely integrate temporal carbon intensity or embodied emissions, and only partially address application mobility, profiling, and runtime lifecycle management.

2.6 Orchestration Platforms for Cloud-Native and Far-Edge Environments

The management of containerized cloud-native applications is tightly coupled with orchestration platforms. Kubernetes [52] has emerged as the de facto cloud operating system for container orchestration. To address constraints in resource-limited or far-edge settings, lightweight distributions such as MicroK8s [53] and K3s [54] have been developed. Zhang et al. [55] propose KOLE, an architecture that scales far-edge nodes by adopting a push-based configuration model, where control-plane nodes push configuration updates to data-plane nodes. This contrasts with Kubernetes' standard pull model and alleviates pressure on control-plane components in large-scale deployments.

These orchestration works primarily target scalability and manageability, with limited direct integration of energy or carbon objectives. They nonetheless provide the substrate on which energy- and carbon-aware controllers, such as those described above or our own proposal, can be implemented.

2.7 Surveys and Taxonomies

Several surveys and mapping studies synthesize energy-aware and fog/edge resource management research. Apat et al. [56] and Bachiega et al. [57] review service placement and resource management strategies in fog and edge computing, identifying limitations in dynamic adaptation, multi-objective optimization, and real-world evaluation. Taleb et al. [58] provide a comprehensive taxonomy and classification of service placement algorithms across integrated cloud–fog/edge environments, highlighting open challenges related to microservices, optimization strategies, and emerging deployment models. Oliveira et al. [31] focus on FaaS in the cloud-to-thing continuum, as discussed earlier, while GreenScale [33] and related work emphasize the need for carbon-aware design. These surveys confirm that, despite extensive attention to energy and latency, the integration of carbon-awareness, multi-tier orchestration, and lifecycle management of microservices remains incomplete.

2.8 Comparison with Our Work

Table 1 compares representative solutions along key dimensions relevant to our work: support for application mobility (migration and placement changes based on user location and latency), energy-aware scheduling, multi-tier deployment (cloud, fog, edge, and multi-domain), workload profiling, dynamic adaptation (e.g., autoscaling and condition-aware reconfiguration), and explicit support for serverless workloads. Our approach targets energy-aware application lifecycle management in CECC environments and is designed to combine application mobility, multi-tier deployment, workload profiling, and dynamic adaptation for cloud-native microservices, rather than focusing on a single dimension (e.g., energy-aware offloading or carbon-aware task shifting).

2.9 Gaps and Emerging Trends

From the reviewed literature, several gaps and trends emerge:

- **Measurement vs. orchestration:** Monitoring frameworks (e.g., Kepler [11], carbon-aware Telco monitoring [51]) provide increasingly accurate power and carbon metrics, but most scheduling and placement solutions only partially exploit this information and rarely close the loop with fine-grained, runtime control policies.
- **Energy-aware but not carbon-aware:** The majority of works optimize energy consumption or power usage; only a small subset explicitly optimizes carbon footprint (e.g., [16], [33], [34], [45]), and even these often rely on static or coarse carbon models that overlook temporal and locational variability of grid carbon intensity.
- **Limited application lifecycle management:** Most approaches treat placement or scheduling as a one-shot optimization problem. Few solutions consider the full lifecycle of applications (profiling, placement, scaling, migration, and

decommissioning) under evolving workloads and infrastructure conditions.

- **Partial coverage of the computing continuum:** Many works focus on either cloud-only or fog/edge-only environments, or at best a simple cloud–fog hierarchy. Comprehensive, practical orchestration across cloud, fog, edge, and device layers remains largely unexplored.
- **Task-level rather than microservice-level abstraction:** A significant portion of the literature models workloads as monolithic tasks, DAGs, or VNF chains. Native microservice architectures, with autoscaling, sidecars, and complex communication patterns typical of Kubernetes-based deployments, are only partially captured in recent works [45], [47].
- **Serverless remains under-explored for sustainability:** Serverless/FaaS research has begun to address energy and QoS-aware orchestration [17], [18], [31], [32], but systematic carbon-aware, multi-tier, lifecycle management for serverless workloads across the continuum is still nascent.
- **Workload profiling and prediction are underutilized:** Only a subset of works (e.g., [20], [26], [27]) build explicit predictive models of workload and energy. Rich profiling of application behavior and long-term patterns is rarely integrated into decision-making logic for placement, migration, or scaling.

Motivated by these gaps, our work aims to provide an energy-aware application lifecycle management framework for cloud–edge computing continuum environments that (i) leverages workload profiling, (ii) supports application mobility and multi-tier deployment, and (iii) enables dynamic adaptation of microservice-based applications under energy and carbon constraints.

3 Application Profiling

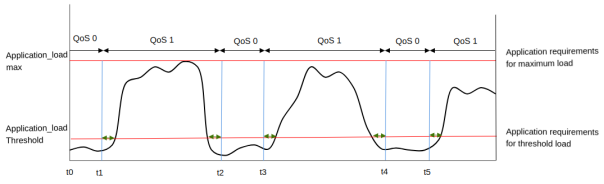


Fig. 1. Proposed application profile

Application profiling has gained interest in recent years. It aims to provide knowledge about the application behavior in different environments. This behavior can be affected by several factors, such as the allocated resources and the load on the application. In our work, we focus on profiling the application resources, which means the amount of computing and networking resources required by the application now and in the near future. This profiling can be done in several ways, and many works tackled the challenge of predicting the load on the application, the resources needed by the application to handle a certain load using both statistics and AI methods, and benchmarking systems. The main statistical methods include (1) the Auto-regressive Integrated Moving Average (ARIMA), which is a combination of the autoregressive model and the moving average model with the goal of predicting future values of a time series; (2) Exponential Smoothing is a time series forecasting method that assigns exponentially decreasing weights over time to past observations; (3) Long short-term memory (LSTM) is a deep neural network that memorizes long-term dependencies of time series data.

We start by describing some of these works, mainly those proposing models to predict application load based on historical trends. Afterward, we will present the application profile model that we propose and how it can be used to achieve more efficient utilization of CECC resources. A first work published in [59] proposes a hybrid model that combines ARIMA with triple exponential smoothing to accurately predict linear and non-linear relationships in the time series of resource workload of Docker containers. The authors claim that the proposed hybrid model improves prediction accuracy over ARIMA or triple exponential smoothing when they are used alone. The system aims to be used directly by users to improve resource utilization by predicting the resources required by containerized workload and scaling the resources accordingly. On the other hand, and focusing on workload prediction, the authors of [60] propose CloudInsight, an online workload prediction framework that addresses dynamic and highly variable cloud workloads. CloudInsight relies on a number of local predictors based on ARIMA, Linear Support Vector Machine (L-SVM), and linear regression, to name a few, and dynamically determines the weights of each local predictor in order to provide a more accurate prediction. Based on the results, the authors claim that the framework can be used to predict real-world cloud workloads. Finally, on empirical profiling methods, work in [61] proposes a framework to obtain fine-grained resource requirements depending on workload characteristics. More concretely, the framework proposes a testing deployment, allowing the capture of the behavior of the application in terms of used resources under varying loads. The observed resource usage can be used afterward to profile the application; the profile would have the number of resources needed to handle certain workloads.

Back to our proposed application profile model,

illustrated by Fig. 1, an application would have a future load; this load can be predicted using the methods described in the existing works that we cited a few of them. The predicted application load could be divided into 1) load below a threshold and 2) load over the threshold. Then, for each case, we define the application requirements that allow the application to handle the maximum load in that period. For instance, in the example illustration, between [$t0$ and $t1$] and [$t2$ and $t3$] and [$t4$ and $t5$], the load on the application is predicted to be below the threshold load. During these periods, the application requirements that allow the application to handle the threshold load are sufficient to handle all the coming workloads to the application since the load is always lower than the threshold. Similarly, between [$t1$ and $t2$] and [$t3$ and $t4$] and [$t5$ and t_{end}], the load is higher than the threshold, meaning that the application requirements that allow the application to handle the threshold load is not sufficient to handle the present load. Consequently, the resources allocated to the application should be increased to meet the requirements of handling the maximum load.

From here, we define the application profile as the following: during the time between $t0$ and t_{end}, the application will have two sets of requirements that we can call QoS (Quality of Service) requirements. The first one is the normal QoS of the application, which allows it to handle all the incoming workloads that we denote QoS_{max}. The second QoS in the resources requirements that are sufficient to handle the threshold load, we denote it QoS_{th}. The application profile should also contain the timestamps at which the application requirement changes. For instance, it will have $times = [t0,\ t1,\ t2,\ t3,\ t4,\ t5,\ t_{end}]$ and $req = [QoS_{th},\ QoS_{max}]$. The times can be either minutes or hours, granular based on the use cases. For instance, a week contains 168 hours or 10080 minutes; this granularity will affect the complexity of the scheduling model described in section 5.

One of the most important elements of this approach is the choice of the threshold load, based on which we will change the application requirements. For this end, we propose a method based on the compute of the area $\mathcal{S}$ covered by the rectangles shaped by QoS_{th} or QoS_{max}, the timestamps of the change in QoS and the y axis as shown in Fig. 2 by the green and orange rectangles. The area covered by those rectangles represents the resource usage over time for each chosen threshold. In Fig. 2, the change of the selected threshold load produces a change in the QoS_{th}, so one way to determine if QoS_{th} or QoS'_{th} is more resources efficient is by comparing the areas $\mathcal{S}$ and $\mathcal{S}'$ and chose the threshold producing the minimum area. Finally, given the continuous nature of the load and requested resources, we can choose discrete values of load collected from profiling frameworks like the one described in [61]. As a matter of course, the proposed model can be scaled to have N cascading thresholds with N different application requirements; in our work, we stick with one threshold.

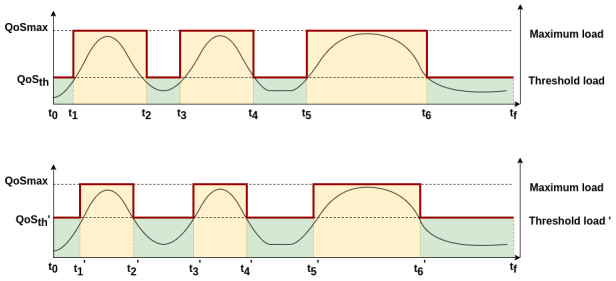


Fig. 2. choice of the load threshold

4 Design and Specification of the Proposed Application Scheduling Framework

4.1 Architecture

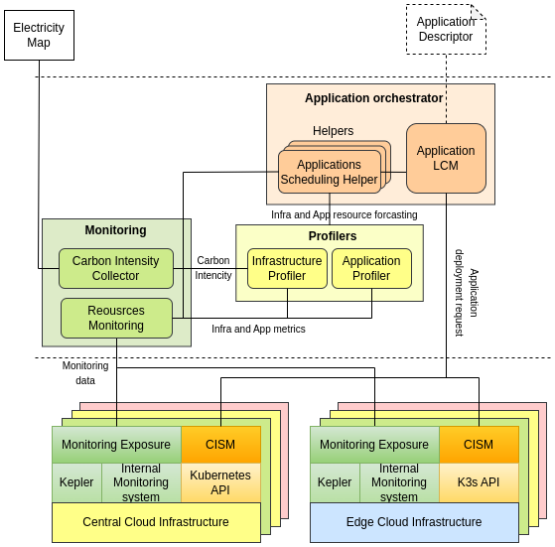


Fig. 3. The overall architecture of the Cloud Edge Computing Continuum Applications Orchestrator

This work aims to provide a resource-efficient application scheduling on top of the Cloud Edge Computing Continuum (CECC) Infrastructures. We consider that CECC spans multiple regions and infrastructures, ranging from cloud infrastructure, including public and private cloud, to edge cloud infrastructure, including far edge devices. Each of these infrastructures is able to run containerized microservices, meaning that it contains a cluster, mainly a Kubernetes cluster or one of its variants, such as the lightweight

Kubernetes K3s. These clusters are managed by a Container Infrastructure Service Manager (CISM); as described in [62], the CISM is the cloud-native equivalent of a Virtualized Infrastructure Manager (VIM), according to ETSI cloud-native report [63]; it is responsible for maintaining the containerized workloads. Moreover, each infrastructure exposes monitoring metrics on the infrastructure and the running applications. Those metrics can be provided by internal monitoring systems such as Prometheus [64], which provide metrics on resource usage, applications, and networks via the multiple metrics exporters that it can collect metrics from. Metrics related to energy used by the running containers and infrastructure metrics can be collected using Kepler. Kepler Exporter exposes a variety of metrics about the energy consumption of Kubernetes components such as Pods and Nodes. Those metrics include:

- **kepler container core joules total:** This metric measures the total energy consumption on CPU cores that a certain container has used.
- **kepler container dram joules total:** This metric represents the total energy spent in DRAM memory by a container.
- **kepler node core joules total:** This metric measures the total energy consumption on CPU cores that a certain node has used.
- **kepler node dram joules total:** This metric represents the total energy spent in DRAM memory by a node.
- **kepler node platform joules total:** This metric represents the total energy consumption of the host.

The above measures are used to provide an energy profiling of the infrastructure. For instance, **kepler_container_core_joules_total**, which provides the energy consumed while using CPU resources in Joules, combined with the container's CPU usage metrics, is used to determine the energy cost of CPU resources at a given infrastructure. So, from these metrics, we can estimate the energy that will be used by the required CPU resources of a container over a period of time. Similarly, we can profile the energy cost of memory resources for a given infrastructure.

The monitoring system collects the metrics from each infrastructure's monitoring exposure endpoints and presents them to the Application Orchestrator, the Application Profiler, and the Infrastructure Profiler. The Monitoring system also contains a Carbon Intensity collector, which collects information about the carbon intensity of the electricity used by the different infrastructures.

Carbon intensity, measured in gCO2eq/kWh, provides the equivalent amount of CO2, which would have the same warming effect on the Earth as the combination of emitted gases. Emitting one gCO2eq is the equivalent of emitting one gram of CO2 in terms of the warming effect that is caused. We use this metric to indicate the quality of energy used by the different CECC infrastructures. Hence, it is preferable to use infrastructures that use clean energy to run the most resource-intensive microservices if those nodes support the required application's QoS. Carbon intensity can be provided as metadata by the infrastructure owner when registering the clusters and nodes to be managed by the framework. If it is not provided, it can be collected from available APIs based on the geographical location of the nodes. For instance, in Europe, Electricity Map [65] provides API endpoints to retrieve the last known carbon intensity (in gCO2eq/kWh) of electricity consumed in an area. Also, National Grid ESO's Carbon Intensity API [66] provides an indicative trend of the regional carbon intensity of the electricity system in Great Britain 96+ hours ahead of real-time. It provides programmatic access to both forecast and estimated carbon intensity data. This measure allows the tracking of the carbon intensity of all infrastructures over time. Furthermore, we propose a carbon intensity collector that will collect the carbon intensity of the electricity used by the managed CECC infrastructures based on the location of the nodes. For this reason, we require the location as part of the information about the infrastructures and their nodes. The carbon intensity collector can collect metrics from different sources based on the regions managed by the application management framework.

The applications profiler is the entity responsible for creating the application profile based on the model described in section 3. It used the metrics from the monitoring system in order to analyse the load on the application and its resource utilization. Afterward, the application profiler selects the 2 levels of requirements QoS_{th} and QoS_{max} as well as the timestamps in the future at which the application will require the respective level of QoS. Indeed, for the newly deployed applications, the profiler needs time to construct the profile; during the profiling period, the requirement entered by the application owner via the application descriptor will be used at all times.

In our system, we consider two types of infrastructure: permanent and temporary. Permanent infrastructures are available for the whole duration of the application execution; we can consider public cloud clusters, sets of edge cloud clusters running on top of machines running continuously without battery or mobility limitations. On the other hand, temporary infrastructures are typically clusters deployed on mobile hosts (e.g., drones), hosts that have battery limitations (e.g., mobile devices), and edge AI single-board computers. For each infrastructure, the profiler contains its type, location, the times at which the infrastructure

resources are available, and the duration of availability. The profile also includes information about the available resources at each infrastructure as well as the network latency to end devices that may communicate with the deployed applications. The profiler is also responsible for concluding the energy usage rate of the infrastructure resources and, based on the location, the carbon intensity of the infrastructure.

Moving upwards, the Application Orchestrator Manages the life cycle of the applications. It is agnostic to the CISM type and communicates with the latter via the CISM plugin. The Application Orchestrator contains the applications scheduling model, which collects information about the application requirements from the applications profiler, including trends and predicted computing and networking resources needed by the application that needs to be scheduled. Then, it uses information about the infrastructure collected from the infrastructure profiler, including the available computing and networking resources, carbon intensity, availability, cost, and energy usage by its different types of resources.Then, the applications scheduling helper, which is part of the application orchestrator, uses this information to provide a plan for the deployment and migration of the application during the deployment period. That is, it will provide the best location to run the application and when the application should moved to other infrastructures using the application requirements, the carbon footprint, as well as the cost of running the overall managed applications.

Finally, the architecture is designed to ensure resiliency at multiple levels. Each cluster embeds its own management applications, thereby localizing failures: if a cluster becomes unavailable, only its CISM and monitoring components are affected, while other clusters remain fully operational. In addition, the profilers and application orchestrator are implemented as containerized, cloud-native microservices. Their stateless design enables replication and deployment across multiple clusters, providing redundancy, mitigating single points of failure.

4.2 Network solutions for seamless migration between the cloud edge and far edge

Since the framework manages the deployment of application micro-services that are deployed at the cloud edge, those application can be migrated based on their profiles representing the requirement of an application. For instance, when Far Edge locations are available near an edge location, which may provide lower latencies for some applications, micro-services of the application can be moved towards these Far Edge locations if low latency is required. While migrating applications, the connectivity between the different micro-services should be continuous, meaning that this migration needs to be seamless. We have investigated techniques to produce seamless migrations between Kubernetes-managed clusters. To ensure this seamless execution of the applications, we should limit the reconfiguration of the microservices after migrating other services that are needed by the application. We could achieve this by providing unchanged network names to application services. This network name is used to reach an application regardless of its location. In the Kubernetes context, this can be achieved using third-party technologies such as Submariner [67], which allows pods in different Kubernetes clusters to communicate seamlessly using secure tunnels to provide connectivity between clusters. Skupper [68] also solves multi-cluster communication challenges by providing a layer 7 service interconnect through Virtual Application Network (VAN). The main advantage of Skupper is that it allows the usage of the same Kubernetes service name to access the application in different clusters. Finally, vxlan (layer 2) can be used in environments where there is a few numbers of nodes such as far edge devices with single node clusters.

Since, in our proposed framework, the applications can be moved from one infrastructure to another based on its requirements, we include the downtime incurred by the migration in the decision-making process of the CECC Application Orchestrator. We performed a test on the migration downtime between an edge computing cluster using Kubernetes and a Far edge cluster using K3s, while the applications are deployed and migrated using a lightweight edge slice orchestration framework presented in [62]. To produce seamless migrations, we use vxlan tunnels to provide connectivity between the Edge and Far edge clusters. For this purpose, we reserve a sub-network of IP addresses at the CISM level to provide it to applications that require connection with other applications that can be moved between the Edge and Far Edge. Then, we create a vxlan tunnel for this sub-network between the Raspberry Pi running K3s and the Intel Tower running Kubernetes. In order to assign an IP address from the reserved pool to the applications pods, we use Container Networking Interfaces such as Multus [69] to create network attachments for pods with the selected static IP address, Calico [70] also can be used to provide static IP to pods.

The results of the migration incurred downtimes are shown in Table. 2.

Those values are used later in section 5 to approximate the downtime caused by the migration of an application and the effect on its availability requirements.

TABLE 2
Measured downtime from the application perspective during migration.

N of apps	Graceful migration downtime (s)	Forced migration downtime (s)
5	0.15	26.41
10	0.17	40.43
15	3.35	50.0
20	0.23	54.48
25	7.32	66.72
30	148.24	136.26
35	93.47	199.28
40	55.93	137.33

5 Application scheduling taking into account the cost and carbon footprint of the deployment

In this section, we formulate the application scheduling into the CECC infrastructure problem via an optimisation problem. The goal of the optimization model is to find the best location to execute applications, knowing that the application can be migrated between the different CECC infrastructures. Fig. 4 presents a visualization of the application deployment; we can see that in our model, we decide where and when the application will be deployed on top of an infrastructure. In order to do so, we introduce the time dimension. The considered timestamps are discrete values extracted from the applications' profile change and the infrastructure availability times. More concretely, the times represent the union of the timestamps from the applications profiles and the timestamps of availability of infrastructure in the case of far edge infrastructures.

For instance, in the example of Fig. 4, the time values are
$[0, 3, 6, 11, 12, 14]$, which is the union of the timestamps from the application profile of app1, app2, app3, and app4 $[0, 6, 14] \cup [0, 3, 6, 12, 14] \cup [0, 6, 14] \cup [0, 6, 11, 14]$.

To do so, we formalise the optimization problem that models the characteristics of the system and applications in order to find the optimal placement of the applications. The used notations are summarized in Table 3.

In the system, we consider a set of CECC infrastructures denoted $\mathcal{C}$ over which applications can be deployed. Each infrastructure c has available resources denoted $\mathcal{R}^t_{c,r}$ at a time $t \in \mathcal{T}$ of resources type $r \in \Gamma$, and network bandwidth $\mathcal{B}^t_c$. On the other hand, we have a set of applications $\mathcal{A}$ that needs to be deployed on top of the managed CECC infrastructures. Each application has a requirements in terms of resources $\rho^t_{a,r}$ of type $r \in \Gamma$ at a time $t \in \mathcal{T}$, bandwidth β^t_a at a time $t \in \mathcal{T}$ and latency to end users λ^t_a at a time $t \in \mathcal{T}$. For each application $a \in \mathcal{A}$, each infrastructure $c \in \mathcal{C}$ provides a latency to the end devices connected to the application represented by $\Lambda^t_{c,a}$. Additionally, each application has an availability requirement denoted δ_a. For example, five-nines availability means that the application should be up and running for 99.999% of the time, while each migration of an application from infrastructure $c1$ to $c2$ produces a downtime of μ_{c_1,c_2}. Further, a QoS breach quota ϵ_a represents the percentage of acceptable time in which the QoS of the application can be breached. The last 2 elements allow the overbooking of the CECC resources without breaching the application requirements. Finally, infrastructures have a cost $\mathcal{M}_c$, a carbon intensity Ψ_c and energy usage for each type of resources $\psi_{c,r}$, which represents the amount of energy consumed by the application when a specific amount of resources is

TABLE 3
Summary of Notations & Variables.

Notation/ Variable	Description
$\mathcal{C}$	Set of CECC infrastructures
$\mathcal{A}$	Set of applications
$\mathcal{T}$	Set of times
Γ	Set of resources
Δ	The total period of the application deployment planning
$\mathcal{W}^t_a$	The set of infrastructures $c \in \mathcal{C}$ that does not satisfy the latency requirements of application $a \in \mathcal{A}$ at time $t \in \mathcal{T}$ meaning $\lambda^t_a < \Lambda^t_{c,a}$
$\mathcal{D}_t$	The duration of the period starting from time $t \in \mathcal{T}$ until time $t+1$
$\mathcal{B}^t_c$	The bandwidth available for the infrastructure $c \in \mathcal{C}$ at time $t \in \mathcal{T}$.
β^t_a	The bandwidth required by application $a \in \mathcal{A}$ at time $t \in \mathcal{T}$.
$\Lambda^t_{c,a}$	The latency from infrastructure $c \in \mathcal{C}$ to the end users of application $a \in \mathcal{A}$ at time $t \in \mathcal{T}$.
λ^t_a	The latency required by application $a \in \mathcal{A}$ at time $t \in \mathcal{T}$.
$\mathcal{R}^t_{c,r}$	The amount of resources of type $r \in \Gamma$ available at CECC infrastructure $c \in \mathcal{C}$ at time $t \in \mathcal{T}$.
$\rho^t_{a,r}$	The amount of resources of type $r \in \Gamma$ required by application $a \in \mathcal{A}$ at time $t \in \mathcal{T}$.
μ_{c_1,c_2}	Migration downtime between two infrastructures $c_1, c_2 \in \mathcal{C}$.
δ_a	The availability required by the application $a \in \mathcal{A}$.
ϵ_a	The maximum allowable duration during which the Quality of Service (QoS) for application $a \in \mathcal{A}$ may be violated.
Ψ_c	The carbon Intensity of infrastructure $c \in \mathcal{C}$.
$\psi_{c,r}$	The energy consumed by a unit of resource of type $r \in \Gamma$.
$\mathcal{M}_c$	The cost of infrastructure $c \in \mathcal{C}$.
$\mathcal{Y}_{a,c}$	A Boolean constant that denotes if application $a \in \mathcal{A}$ was deployed on top of infrastructure $c \in \mathcal{C}$ before the time of placement.
α	A constant that specifies the priority between the cost and carbon footprint.
$\mathcal{X}^t_{a,c}$	A Boolean variable that denotes if the application $a \in \mathcal{A}$ is deployed on top of infrastructure $c \in \mathcal{C}$ at time $t \in \mathcal{T}$.

allocated to it. This metric can be provided by the infrastructure profiler based on the energy metrics collected from Kepler.

The variable of the system is $\mathcal{X}^t_{a,c}$ which represents if the application a is deployed on top of infrastructure c at time t. We also add $\mathcal{Y}_{a,c}$, which specifies if the application a was running on top of infrastructure c before computing the solution of the problem. This variable is used to keep track of the applications that are already running.

Note that the problem definition is intended to be flexible and adapt to the changes on the application profile on runtime, either due to bursts of load on the application that do not follow the application profile or adaptation of the application profile based on new monitoring information. This change accommodation is done using the values of $\mathcal{Y}_{a,c}$, which allows us to solve the problem at a time t^δ using the adapted application requirements by setting its value based on the current infrastructure on top of which the application is running, ϵ_a and δ_a which can be adapted based on the measured downtimes and QoS breach duration that occurred during the planned period.

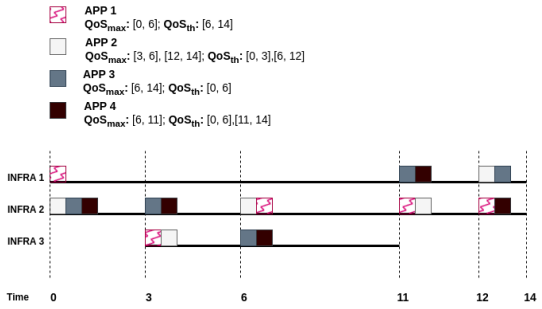


Fig. 4. Dynamic application deployment on the CECC infrastructure over time

5.1 Objective function

Based on the above discussion, the objective of the problem under consideration is twofold: 1) Minimize the cost of deployment, which is based on the type of infrastructure used. 2) Minimize the Carbon footprint of the deployment on top of the large CECC infrastructure, which is based on the type of energy (electricity source) used by the infrastructure. We define the problem as a multi-objective optimization problem, where the objective function is a function of two objectives. Its mathematical formulation is as follows:

$$\mathcal{F}(\mathcal{X}^t_{a,c}) \; = min \; f(\mathcal{F}_1(\mathcal{X}^t_{a,c}),\; \mathcal{F}_2(\mathcal{X}^t_{a,c})) \tag{1}$$

The function, s.t, constraints (4 - 8) presented bellow, represents the multi-objective optimization problem definition for efficient and carbon aware application scheduling on top of different types of CECC infrastructures. with $\mathcal{X}^t_{a,c}$ being the binary decision variable. The variable's value determines whether the application $a \in \mathcal{A}$ is deployed on top of the infrastructure $c \in \mathcal{C}$ at time $t \in \mathcal{T}$ (value 1) or not (value 0).

5.1.1 Cost objective function

The first objective is to minimize the cost of the deployment of the applications. Based on the used resources, each application, when deployed on top of an infrastructure, will produce a cost. In function 2, we calculate the cost incurred by all the applications in the system. This cost is also affected by the time spent by an application on a specific infrastructure. $\mathcal{Z}_1$ is a normalization constant.

$$\mathcal{F}_1(\mathcal{X}^t_{a,c}) = \frac{1}{\mathcal{Z}_1 \times \Delta} \sum_{a\in\mathcal{A}} \sum_{c\in\mathcal{C}} \sum_{t\in\mathcal{T}} \sum_{r\in\Gamma} \mathcal{D}_t \times \mathcal{M}_c \times \rho^t_{a,r} \times \mathcal{X}^t_{a,c} \tag{2}$$

5.1.2 Carbon footprint objective function

The second objective is to minimize the system's carbon footprint. To do so, we use the carbon intensity of the infrastructure multiplied by the resources and energy usage for the same infrastructure. Using the application's required resources, we can estimate the carbon footprint of the application deployment. Function 3 calculates the carbon footprint of the system based on the time spent by the applications at each infrastructure. $\mathcal{Z}_2$ is a normalization constant.

$$\mathcal{F}_2(\mathcal{X}^t_{a,c}) = \frac{1}{\mathcal{Z}_2 \times \Delta} \sum_{a\in\mathcal{A}} \sum_{c\in\mathcal{C}} \sum_{t\in\mathcal{T}} \sum_{r\in\Gamma} \mathcal{D}_t \times \Psi_c \times \psi_{c,r} \times \rho^t_{a,r} \times \mathcal{X}^t_{a,c} \tag{3}$$

5.2 Constraints

The constraints of the optimization problem are the following:

5.2.1 Instances constraints

Constraint 4 is used to make sure that at each time t, the application is running on an infrastructure.

$$\forall a \in \mathcal{A}, \forall t \in \mathcal{T} : \sum_{c\in\mathcal{C}} \mathcal{X}^t_{a,c} = 1 \tag{4}$$

5.2.2 Availability constraints

Constraint 5 aims to make sure that the application is available δ_a of the time. In our system, we consider only the downtime caused by the migration of the application from one infrastructure to another.

$$\forall a \in \mathcal{A} : \sum_{c_1\in\mathcal{C}} \sum_{c_2\in\mathcal{C}} \sum_{t>0\in\mathcal{T}} \frac{\mu_{c_1,c_2}}{\Delta} \times \mathcal{X}^t_{a,c_1} \times \mathcal{X}^{t-1}_{a,c_2} + \sum_{c1\in\mathcal{C}} \sum_{c1\in\mathcal{C}} \frac{\mu_{c_1,c_2}}{\Delta} \times \mathcal{Y}_{a,c_1} \times \mathcal{X}^0_{a,c_2} \leq 1 - \delta_a \tag{5}$$

5.2.3 Latency constraints

Constraint 6 aims to assure that the application's latency requirement is respected for at least $1 - \epsilon_a$ of the time. To do so, we check that the time spent by the application running on infrastructures that do not respect its latency requirements (represented by $\mathcal{W}_a^t$) is less than ϵ_a.

$$\forall a \in \mathcal{A} : \sum_{t \in \mathcal{T}} \sum_{c \in \mathcal{W}_a^t} \frac{\mathcal{D}_t}{\Delta} \times \mathcal{X}_{a,c}^t \leq \epsilon_a \tag{6}$$

5.2.4 bandwidth constraints

We introduce constraint 7 in order to guarantee that each infrastructure can satisfy the bandwidth requirements of the applications running on top of it at every time t.

$$\forall c \in \mathcal{C}, \forall t \in \mathcal{T} : \sum_{a \in \mathcal{A}} (1 - \epsilon_a) \times \beta_a^t \times \mathcal{X}_{a,c}^t \leq \mathcal{B}_c^t \tag{7}$$

5.2.5 resources constraints

Lastly, constraint 8 is introduced to ensure that each infrastructure can satisfy the resource requirements of the applications assigned to it at every time t.

$$\forall c \in \mathcal{C}, \forall r \in \Gamma, \forall t \in \mathcal{T} : \sum_{a \in \mathcal{A}} \rho_{a,r}^t \times \mathcal{X}_{a,c}^t \leq \mathcal{R}_{c,r}^t \tag{8}$$

5.3 Resiliency and adaptation mechanisms

In real-world scenarios, application profiles may evolve over time, while system components such as monitoring modules or external data sources can occasionally fail. To ensure robustness under such conditions, the proposed framework incorporates several resiliency and adaptation mechanisms.

First, the resource allocation model is designed to be re-executed at any point during the application's deployment. This allows the system to adapt dynamically whenever a new profile becomes available or when the existing profile is found to be inaccurate. In cases where profiling information is unreliable, the model falls back to the maximum QoS requirement (QoS_{max}), which serves as a safe baseline to guarantee service continuity. The re-execution process is informed by key parameters, including accumulated downtime, accumulated QoS breaches, and the last placement of the application, as shown in Fig. 5, which guide the optimization to remain consistent with system objectives while avoiding unnecessary migrations. In this way, short-term profiling errors or unexpected variations in energy availability are mitigated through continuous re-profiling and dynamic re-optimization.

Second, to account for external data dependencies, the system implements layered fallback strategies. For

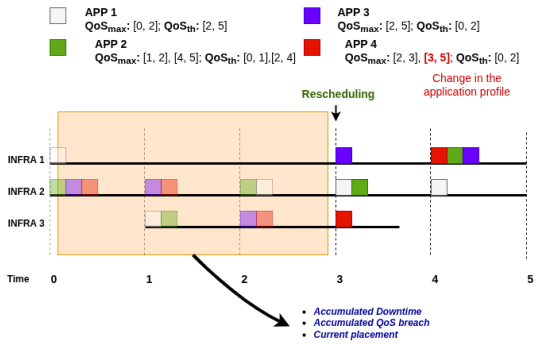


Fig. 5. Adaptation of the application deployment plan due to application profile change

example, if the Electricity Map API becomes unavailable, the system defaults to the last recorded value of the infrastructure's carbon intensity. If no historical data is available, a further fallback approximation is derived from the average carbon intensity of the country where the infrastructure is located. This ensures that the optimization remains functional even in the absence of real-time carbon data.

Finally, in situations where monitoring metrics cannot be collected, the system once again reverts to allocating resources based on QoS_{max}. Since monitoring data is essential to maintain an accurate profile, this conservative fallback strategy provides reliability by safeguarding the application's proper functioning.

Through these mechanisms, the framework achieves resiliency against profiling inaccuracies, monitoring failures, and external data unavailability, ensuring robust and adaptive resource management across diverse deployment conditions.

5.4 Application scheduling algorithm

Obtaining the optimal solution to the proposed problem can be costly in terms of used resources and execution time as shown in section 6. Especially if the number of applications and infrastructure increases, making it unusable in real scenarios in which the system cannot afford to wait hours and use a high number of CPU cores and memory resources to schedule applications. So, in order to solve the problem, we propose algorithm 1; the aim of the algorithm is to provide a heuristic solution to the MOOP 1 in a short time while consuming a limited amount of resources.

First, in the algorithm, we sort the available infrastructures based on their carbon intensity and cost, with the variable α representing the importance of each aspect (line 1). Then as described above, the application profile provides two levels of QoS: QoS_{th} and QoS_{max}. The variable $\mathcal{SP}$ contains all the QoS levels for all applications sorted based on latency requirement (line 6). For each timestamp from the set

Algorithm 1 Application deployment algorithm

1: $\mathcal{SC} \leftarrow$ Sort($\mathcal{C}, asc$) {Sort infrastructure according to cost and carbon intensity using $\frac{\alpha \times \Psi_c}{MaxIntensity} + \frac{(1-\alpha)\times \mathcal{M}_c}{Maxcost}$ value}
2: **for** a **in** $\mathcal{A}$ **do**
3: $\mathcal{P}$.insert($a.QoS_{th}$)
4: $\mathcal{P}$.insert($a.QoS_{max}$)
5: **end for**
6: $\mathcal{SP} \leftarrow$ Sort($\mathcal{P}, asc$) {Sort application profiles based on the required latency}
7: **for** t **in** $\mathcal{T}$ **do**
8: Initialize infrastructure allocatable resources and bandwidth $\mathcal{AL}_c$
9: **for** p **in** $\mathcal{P}$ **do**
10: $a \leftarrow p.app$
11: **if** ProfileActive($p,\ t$) **then**
12: $allocated \leftarrow$ **false**
13: $\gamma \leftarrow Random(0,1)$
14: **if** $\gamma \geq \frac{AllowedDowntime_a \times Random(0,1)}{\delta_a \times \Delta}$ **then**
15: **if** KeepAppOnInfra(p) **then**
16: Keep the application running on the same infrastructure it is running on
17: $allocated \leftarrow$ **true**
18: Update Infra Allocatable Resources $\mathcal{AL}_c$
19: **continue** to next application
20: **end if**
21: **end if**
22: Application can be migrated
23: **for** c **in** $\mathcal{SC}$ **do**
24: **if** DeployAppOnInfra($p,\ c,\ \epsilon_a,\ t$) **then**
25: The application will deployed on infrastructure c at time t
26: $allocated \leftarrow$ **true**
27: Update Infra Allocatable Resources $\mathcal{AL}_c$
28: Update $AllowedDowntime_a$
29: **continue** to next application
30: **end if**
31: **end for**
32: **end if**
33: **end for**
34: **end for**

of times, the algorithm maps each application to an infrastructure. To do so, we define a random variable γ that helps decide if the application will keep running on the current infrastructure, on top of which it was running for $t-1$ (if the infrastructure has enough resources to accommodate the application requirements). Otherwise, the ordered list of infrastructures will be explored to find the first fitting infrastructure that satisfies the application requirements at time t. After each assignment of an application to an infrastructure, the available infrastructure resources as well as the application $AllowedDowntime_a$ are updated (lines 18, 27, 28).

The function KeepAppOnInfra(p) checks if the application a with QoS p can be kept running on top of the infrastructure over which it was running at time $t-1$. While DeployAppOnInfra($p,\ c,\ \epsilon_a,\ t$) checks if the application a can with QoS p can be deployed on top of infrastructure c at time t, if so it updates the application's ϵ_a value.

6 Results

To solve the optimization problem, we implemented a simulator for applications, applications' profiles and infrastructures. The simulator was used to create scenarios with a varying number of applications, infrastructure, and duration. Based on the required configuration, which includes the number of applications, the number of infrastructures and far edge infrastructures. The simulator produces several scenarios based on: the desired configuration, which includes the number of applications, the required application availability, the number of infrastructures (including far-edge infrastructures), and the deployment duration. These scenarios include the application requirements over time, the latency between each infrastructure and the application's users, the available resources at each infrastructure, the infrastructure availability period, and the associated carbon intensity. In addition, it provides the estimated migration time between different infrastructures.

We implemented our simulation environment using Python and used Gurobi version 11.0.0 to get the optimal solutions for the test use cases. The tests are done on top of an Intel server PowerEdge T440 with 128GB of RAM and 64 Core (Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz) with hyperthreading enabled. Using Ubuntu 20.04.6 LTS an operating system.

We solve the model using α scalarization (equation 9)

$$min\ \alpha \times \mathcal{F}_1(\mathcal{X}^t_{a,c}) + (1-\alpha) \times \mathcal{F}_2(\mathcal{X}^t_{a,c}) \qquad (9)$$

with:

- $\alpha = 0.5$ to get the optimal solution with a trade-off between minimizing cost and carbon footprint,
- $\alpha = 0$ to get the optimal solution for carbon footprint minimization
- And $\alpha = 1$ to get the optimal solution for cost minimization.
- We also solve the problem using $\alpha = 0.1$ representing the carbon footprint-focused model.
- Wile using $\alpha = 0.9$ we solve the cost focused model.

Similarly, we test the heuristic solution using different trade-offs. The trade-offs can be configured using the values of $\frac{\alpha \times \Psi_c}{MaxIntensity} + \frac{(1-\alpha) \times \mathcal{M}_c}{Maxcost}$ where:

- we use $\alpha = 0.5$ while ordering infrastructures to get the balanced heuristic solution.
- $\alpha = 0.1$ to get the cost-focused heuristic solution.
- And $\alpha = 0.9$ to get the carbon footprint-focused heuristic solution.

6.1 Test results comparison

In order to test the model's performance, we design several scenarios of deployments and infrastructure states. In each scenario, we specify the number of applications to be deployed, the number of infrastructures available, and the duration of the period over which the model plans the application LCM procedures. The scenarios range from deploying 10 applications on top of 4 infrastructures to deploying 100 applications on top of 40 infrastructures. For each scenario, we consider that 10% of infrastructure is composed of mobile far-edge infrastructures, meaning that they are available only for a period of the whole duration of deployment planning. We also vary the test duration, representing the period for which we plan the deployment of the application. We start from 10 to 100 and finally, a duration of 1000. The duration represents the period over which we plan to run the application. Note that the period can be periodic, meaning that at the end of the duration, we plan the deployment of the application over the next duration again while taking into account the placement of the applications at the end of the last duration, which is essential in order to satisfy the availability constraints of the applications. In each scenario, the Infrastructure profile is represented by: the availability of the infrastructure (permanent or temporary and the time of availability), the available compute resources, the available network bandwidth, the latency from the infrastructure to the target user of each application, the carbon intensity and the energy consumption of its resources. Moreover, each application has a profile that follows the same model presented in Section 3 containing the required compute and network resources for each period of the duration of the deployment.

For the first results (Figs. 6, 7), we evaluate the performance gap among various models: The Carbon Footprint Gurobi Optimization Model demonstrates the optimal carbon footprint achievable in each scenario. The Cost Gurobi Optimization Model shows the optimal cost attainable in each scenario. Additionally, the Trade-off Gurobi Optimization Models (comprising balanced configuration, cost-focused, and carbon footprint-focused configurations) are analyzed. Finally, the proposed heuristic uses the balanced configuration ($\alpha = 0.5$).

Fig. 6 illustrates the gap between the optimal carbon footprint that can be achieved in each scenario and the carbon footprint of the carbon footprint-focused Gurobi model, the heuristic carbon footprint, and the balanced Gurobi model. We notice from the obtained results that the carbon footprint-focused Gurobi model is consistently the closest in terms of carbon efficiency to the optimal solution with a gap of less than 25%. Meanwhile, the heuristic carbon efficiency is similar to the balanced Gurobi model, as we can notice the heuristic performed better in the scenarios: (apps:30, duration: 10), (apps:40, duration: 100), (apps:50, duration: 100), (apps:70, duration: 100), (apps:10, duration: 1000) and (apps:60, duration: 1000).

Similarly, Fig. 7 shows the gap between the optimal cost that can be achieved in each scenario and the cost of the cost-focused Gurobi model, the heuristic cost, and the balanced Gurobi model. Again, we notice from the obtained results that the cost-focused Gurobi model is consistently the closest in terms of cost efficiency to the optimal solution with a gap of less than 50%. While the heuristic cost efficiency is similar to the balanced Gurobi model in some scenarios, it fell short in other scenarios with a gap between the 2 models being higher than 90%. However, we can notice that the heuristic performed better in the scenarios: (apps:50, duration: 10), (apps:60, duration: 10), (apps:10, duration: 100), (apps:90, duration: 100), (apps:10, duration: 1000) and (apps:70, duration: 1000).

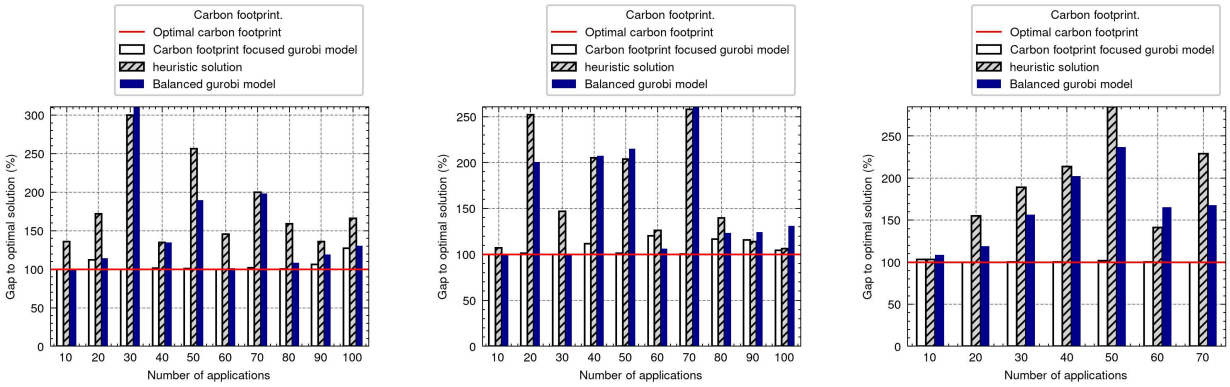
We also examine how altering the heuristic configuration impacts the results. In Figs. 8, 9 we measure the cost and carbon efficiency of the Carbon footprint-focused heuristic solution, cost-focused heuristic solution, and the balanced heuristic solution.

From Fig. 8, which depicts the gap in terms of carbon efficiency between the models, we notice that changing the configuration of the heuristic by giving higher weight to the carbon footprint results in better carbon efficiency as we notice that the gap between carbon footprint-focused heuristic and the optimal solution is less than 40% except in 4 scenarios: (apps:20, duration: 10), (apps:20, duration: 100), (apps:40, duration: 100) and (apps:50, duration: 1000).

In Fig. 9, we can see that the gap in terms of cost efficiency between the cost-focused heuristic and the optimal cost solution is consistently less than 50% except in 7 scenarios: (apps:30, duration: 10), (apps:20, duration: 100), (apps:40, duration: 100), (apps:20, duration: 1000), (apps:50, duration: 1000), (apps:60, duration: 1000).

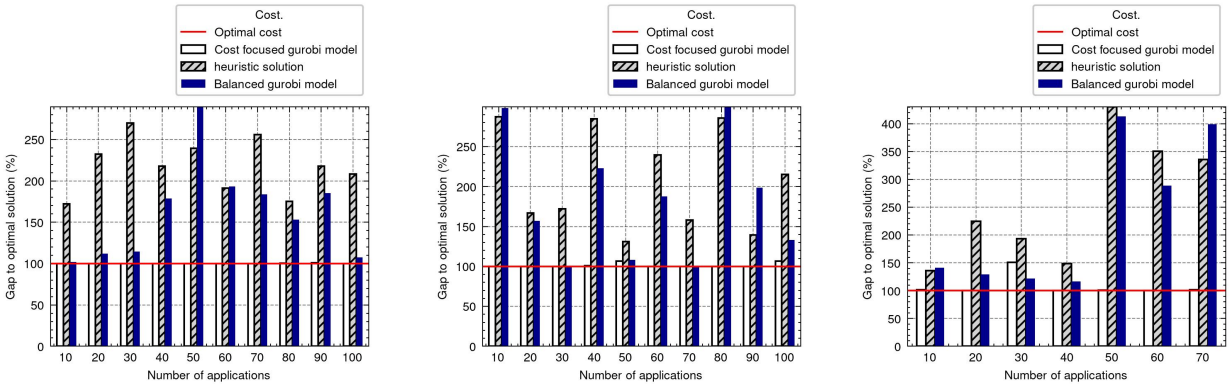
6.2 Discussion

From the results obtained, we can conclude that using the heuristic method reduces the time and resources



(a) Duration : 10 (b) Duration : 100 (c) Duration : 1000

Fig. 6. The gap in Carbon Footprint (%) produced by the models for a duration of 10, 100, and 1000



(a) Duration : 10 (b) Duration : 100 (c) Duration : 1000

Fig. 7. The gap in the Cost (%) produced by the models for a duration of 10, 100 and 1000

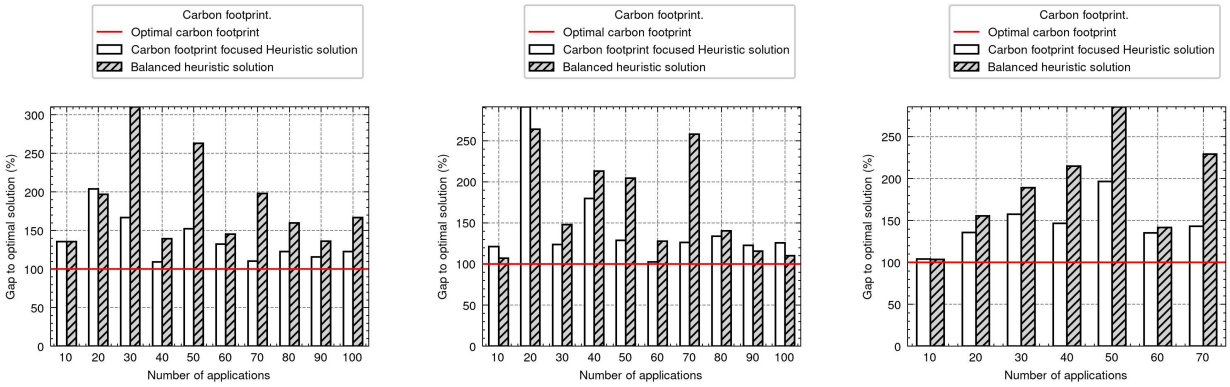
needed to plan the placement and potential migration of the deployed applications. Furthermore, it provides results that are comparable to the optimal cost or carbon efficiency that can be achieved. During the construction of the test scenarios, we did not establish a relationship between the cost of an infrastructure and its carbon intensity. This can be seen in the results, where there is a dispersion between the performance of the different models. From the results obtained, we believe that the granularity of the decision model should be configurable depending on the needs of each deployment. Indeed, for different types of applications, the deployment priority can change. From a CECCM operator point of view, the cost minimization of workloads may have a higher priority than minimizing the carbon footprint of the overall CECC deployments. The decision can also be made by the application owner, who can require that his application's carbon footprint be minimized.

7 Conclusion

In this paper, we introduced a CECC management framework. We provided a method for defining an application profile in a way that can be used for application deployment and migration. The CECC application orchestrator also uses the infrastructure profile, which includes the carbon intensity, energy usage of its resources, and the availability of the infrastructure. We proposed a mathematical problem definition for planning the deployment/migration of applications while satisfying their availability, compute resources and network requirements. Afterward, we solved the problem for different deployment scenarios.

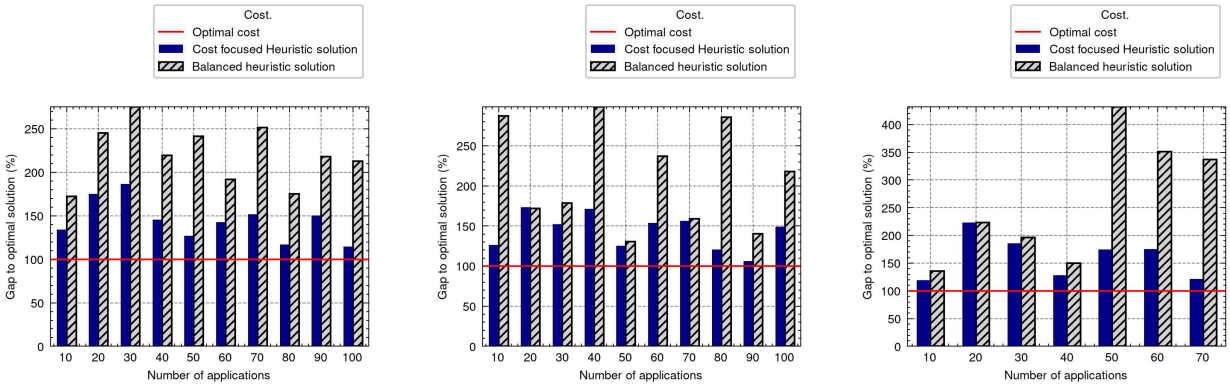
Acknowledgment

This work was partially supported by Agence Nationale de la Recherche (ANR) under the project NF-MUST (Grant Id. 22-PEFT-0002)



(a) Duration : 10 (b) Duration : 100 (c) Duration : 1000

Fig. 8. The gap in Carbon Footprint (%) produced by the different heuristic configurations for a duration of 10, 100 and 1000



(a) Duration : 10 (b) Duration : 100 (c) Duration : 1000

Fig. 9. The gap in the Cost (%) produced by the different heuristic configurations for a duration of 10, 100 and 1000

REFERENCES

[1] A. Boutouchent, K. Boutiba, and A. Ksentini, "Budget-aware resource pricing in cloud and edge computing continuum," in *2024 20th International Conference on Network and Service Management (CNSM)*, 2024.

[2] A. E. Meliani, M. Mekki, and A. Ksentini, "Resiliency focused proactive lifecycle management for stateful microservices in multi-cluster containerized environments," *Comput. Commun.*, 2025.

[3] N. Toumi, M. Bagaa, and A. Ksentini, "Machine learning for service migration: A survey," *IEEE Communications Surveys Tutorials*, 2023.

[4] A. Aissioui, A. Ksentini, A. M. Gueroui, and T. Taleb, "On enabling 5g automotive systems using follow me edge-cloud concept," *IEEE Transactions on Vehicular Technology*, 2018.

[5] A. E. Meliani, A. Ksentini, M. Mekki, A. Kadouma, D. Amaxilatis, A. Ba, E. Ojeda Coronado, J. Beredimas, V. Mouloss, S. Sengupta, D. Klonidis, and C. Verikoukis, "Ai-native cecc management architecture: Enabling scalable and intelligent cloud-edge computing," in *In proceeding of SDN-NFV conference*, IEEE, Ed., Athens, 2025.

[6] M. Mekki, N. Toumi, and A. Ksentini, "Microservices configurations and the impact on the performance in cloud native environments," in *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, 2022.

[7] M. A. Elghani, A. Sagar, A. Ksentini, and R. Knopp, "Timetrack: A dataset for exploring temporal patterns and predictive insights into openairinterface (oai) ci/cd cluster," in *ICC 2025 - IEEE International Conference on Communications*, 2025.

[8] M. Mekki, S. Arora, and A. Ksentini, "A scalable monitoring framework for network slicing in 5g and beyond mobile networks," *IEEE Transactions on Network and Service Management*, 2022.

[9] T. E. G. Deal. (June 2024). [Online]. Available: https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/european-green-deal_en

[10] B. Knowles, "Acm techbrief: Computing and climate change," 2021.

[11] K. E. P. L. Exporter. (June 2024). [Online]. Available: https://sustainable-computing.io

[12] K. Kaur, S. Garg, G. Kaddoum, S. H. Ahmed, and M. Atiquzzaman, "Keids: Kubernetes-based energy and interference driven scheduler for industrial iot in edge-cloud ecosystem," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4228–4237, 2020.

[13] S. Tripathi, P. Kumar, P. Gupta, R. Misra, and T. Singh, "Workload shifting based on low carbon intensity periods: A framework for reducing carbon emissions in cloud computing," in *2023 IEEE International Conference on Big Data (BigData)*, 2023, pp. 3387–3395.

[14] S. Gupta and A. Gupta, "Optimising the carbon footprint

for cloud resources in a cloud environment," in *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*, 2024, pp. 215–221.

[15] H. R. Chi, D. Corujo, and R. L. Aguiar, "Carbon-aware full-decentralized multi-provider edge computing peer offloading," in *2024 IEEE 22nd International Conference on Industrial Informatics (INDIN)*, 2024, pp. 1–6.

[16] Z. Yu, Y. Zhao, T. Deng, L. You, and D. Yuan, "Less carbon footprint in edge computing by joint task offloading and energy sharing," *IEEE Networking Letters*, vol. 5, no. 4, pp. 245–249, 2023.

[17] M. S. Aslanpour, A. N. Toosi, M. A. Cheema, and R. Gaire, "Energy-aware resource scheduling for serverless edge computing," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2022, pp. 190–199.

[18] S. Qi, H. Moore, N. Hogade, D. Milojicic, C. Bash, and S. Pasricha, "Casa: A framework for slo- and carbon-aware autoscaling and scheduling in serverless cloud computing," in *2024 IEEE 15th International Green and Sustainable Computing Conference (IGSC)*, 2024, pp. 1–6.

[19] U. Vadde and V. S. Kompalli, "Energy efficient service placement in fog computing," *PeerJ Computer Science*, vol. 8, p. e1035, 2022.

[20] R. K. Naha, S. Garg, S. K. Battula, M. B. Amin, and D. Georgakopoulos, "Multiple linear regression-based energy-aware resource allocation in the fog computing environment," *Computer Networks*, vol. 216, p. 109240, 2022.

[21] A. Bozorgchenani, D. Tarchi, and G. E. Corazza, "Centralized and distributed architectures for energy and delay efficient fog network-based edge computing services," *IEEE Transactions on Green Communications and Networking*, vol. 3, no. 1, pp. 250–263, 2019.

[22] K. Gai, X. Qin, and L. Zhu, "An energy-aware high performance task allocation strategy in heterogeneous fog computing environments," *IEEE Transactions on Computers*, vol. 70, no. 4, pp. 626–639, 2021.

[23] W. N. W. Muhamad, S. S. Mohd Aris, K. Dimyati, M. A. Javed, I. A. Rai, D. Mohd Ali, and E. Abdullah, "Energy-efficient task offloading in fog computing for 5G cellular network," *Engineering Science and Technology, an International Journal*, vol. 50, p. 101628, 2024.

[24] M. A. Alshahrani, A. A. Qidan, T. E. H. El-Gorashi, and J. M. H. Elmirghani, "Energy efficient service placement for IoT networks," in *2024 24th International Conference on Transparent Optical Networks (ICTON)*, 2024, pp. 1–5.

[25] S. Azizi, M. Shojafar, J. Abawajy, and R. Buyya, "Deadline-aware and energy-efficient IoT task scheduling in fog computing systems: A semi-greedy approach," *Journal of Network and Computer Applications*, vol. 201, p. 103333, 2022.

[26] C. Gu *et al.*, "Energy efficient task allocation and energy scheduling in green energy powered edge computing," *Future Generation Computer Systems*, vol. 95, pp. 89–104, 2019.

[27] R. Ullah, M. Yahya, L. Mostarda, A. Alshammari, A. I. Alutaibi, N. Sarwar, F. Ullah, and S. Ullah, "Intelligent decision making for energy efficient fog nodes selection and smart switching in the IoT: A machine learning approach," *PeerJ Computer Science*, vol. 10, p. e1833, 2024.

[28] A. Masri and M. Al-Jabi, "Toward IoT fog computing-enabled system energy consumption modeling and optimization by adaptive TCP/IP protocol," *PeerJ Computer Science*, vol. 7, p. e653, 2021.

[29] R. A. Alsemmeari, M. Y. Dahab, B. Alturki, A. A. Alsulami, and R. Alsini, "Towards an effective service allocation in fog computing," *Sensors*, vol. 23, no. 17, p. 7327, 2023.

[30] A. J. Roumeliotis, E. Myritzis, E. Kosmatos, K. V. Katsaros, and A. J. Amditis, "Multi-area, multi-service and multi-tier edge-cloud continuum planning," *Sensors*, vol. 25, no. 13, p. 3949, 2025.

[31] B. da Silva Oliveira, N. Ferry, H. Song, R. Dautov, A. Barisic, and A. Rego da Rocha, "Function-as-a-service for the cloud-to-thing continuum: A systematic mapping study," in *Proceedings of the 8th International Conference on Internet of Things, Big Data and Security (IoTBDS)*. SCITEPRESS, 2023, pp. 82–93.

[32] N. Filinis, I. Tzanettis, D. Spatharakis, E. Fotopoulou, I. Dimolitsas, A. Zafeiropoulos, C. Vassilakis, and S. Papavassiliou, "Intent-driven orchestration of serverless applications in the computing continuum," *Future Generation Computer Systems*, vol. 154, pp. 72–86, 2024.

[33] Y. G. Kim, U. Gupta, A. McCrabb, Y. Son, V. Bertacco, D. Brooks, and C.-J. Wu, "Greenscale: Carbon-aware systems for edge computing," *CoRR*, vol. abs/2304.00404, 2023.

[34] E. Ahvar, S. Ahvar, Z. Ádám Mann, N. Crespi, R. Glitho, and J. Garcia-Alfaro, "Deca: A dynamic energy cost and carbon emission-efficient application placement method for edge clouds," *IEEE Access*, 2021.

[35] E. Carlini, M. Coppola, P. Dazzi, and L. Ricci, "Self-optimising decentralised service placement in heterogeneous cloud federation," in *Proc. IEEE SASO*, 2016.

[36] M. Chen, W. Li, G. Fortino, Y. Hao, L. Hu, and I. Humar, "A dynamic service migration mechanism in edge cognitive computing," *ACM Trans. Internet Technol.*, 2019.

[37] T. Djemai, P. Stolf, T. Monteil, and J.-M. Pierson, "A discrete particle swarm optimization approach for energy-efficient iot services placement over fog infrastructures," in *Proc. ISPDC*, 2019.

[38] H. O. Hassan, S. Azizi, and M. Shojafar, "Priority, network and energy-aware placement of iot-based application services in fog-cloud environments," *IET Communications*, 2020.

[39] Y. Li and S. Wang, "An energy-aware edge server placement algorithm in mobile edge computing," in *Proc. IEEE EDGE*, 2018.

[40] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi, "Multiobjective optimization for computation offloading in fog computing," *IEEE Internet Things J.*, 2018.

[41] M. G. Mortazavi, M. H. Shirvani, and A. Dana, "A discrete cuckoo search algorithm for reliability-aware energy-efficient iot applications multi-service deployment in fog environment," in *Proc. ICECET*, 2022.

[42] B. V. Natesha and R. M. R. Guddeti, "Adopting elitism-based genetic algorithm for minimizing multi-objective problems of iot service placement in fog computing environment," *Journal of Network and Computer Applications*, 2021.

[43] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *Proc. IEEE INFOCOM*, 2019.

[44] K. H. K. Reddy, A. K. Luhach, B. Pradhan, J. K. Dash, and D. S. Roy, "A genetic algorithm for energy efficient fog layer resource management in context-aware smart cities," *Sustainable Cities and Society*, 2020.

[45] A. Saboor, A. K. Mahmood, A. H. Omar, M. F. Hassan, S. N. M. Shah, and A. Ahmadian, "Enabling rank-based distribution of microservices among containers for green cloud computing environment," *Peer-to-Peer Networking and Applications*, 2022.

[46] M. Shokouhifar, "Fh-aco: Fuzzy heuristic-based ant colony optimization for joint virtual network function placement and routing," *Applied Soft Computing*, 2021.

[47] I. Taleb, J.-L. Guillaume, and B. Duthil, "Energy- and resource-aware graph-based microservices placement in the cloud-fog-edge continuum," in *Proc. ICCS*, 2024.

[48] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Transactions on Services Computing*, 2018.

[49] H. H. A. Valera, M. Dalmau, P. Roose, J. Larracoechea, and C. Herzog, "Draceo: A smart simulator to deploy energy

saving methods in microservices based networks," in *Proc. WETICE*, 2020.
[50] Y. Yu, J. Yang, C. Guo, H. Zheng, and J. He, "Joint optimization of service request routing and instance placement in the microservice system," *Journal of Network and Computer Applications*, 2019.
[51] M.-Y. Wu, W.-T. Su, C.-J. Tsai, C.-W. Hsieh, C.-H. Lee, and S.-H. Hsu, "Carbon emission monitoring for telco cloud," in *2023 IEEE 15th International Conference on Computational Intelligence and Communication Networks (CICN)*, 2023, pp. 93–98.
[52] Kubernetes. (June 2024). [Online]. Available: https://kubernetes.io/
[53] Microk8s. [Online]. Available: https://microk8s.io/
[54] K3s. [Online]. Available: https://k3s.io/
[55] J. Zhang, C. Jin, Y. Huang, L. Yi, Y. Ding, and F. Guo, "Kole: Breaking the scalability barrier for managing far edge nodes in cloud," in *Proceedings of the 13th Symposium on Cloud Computing*, 2022, pp. 196–209.
[56] H. K. Apat, V. Goswami, B. Sahoo, R. K. Barik, and M. J. Saikia, "Fog service placement optimization: A survey of state-of-the-art strategies and techniques," *Computers*, vol. 14, no. 3, p. 99, 2025.
[57] J. Bachiega Jr, B. G. S. Costa, L. R. Carvalho, M. J. Rosa, and A. Araujo, "Computational resource allocation in fog computing: A comprehensive survey," *ACM Computing Surveys*, vol. 55, no. 14s, pp. 1–38, 2023.
[58] I. Taleb, J.-L. Guillaume, and B. Duthil, "A survey on services placement algorithms in integrated cloud-fog / edge computing," *ACM Comput. Surv.*, vol. 57, no. 11, Jun. 2025. [Online]. Available: https://doi.org/10.1145/3729214
[59] Y. Xie, M. Jin, Z. Zou, G. Xu, D. Feng, W. Liu, and D. Long, "Real-time prediction of docker container resource load based on a hybrid model of arima and triple exponential smoothing," *IEEE Transactions on Cloud Computing*, vol. 10, no. 2, pp. 1386–1401, 2022.
[60] I. K. Kim, W. Wang, Y. Qi, and M. Humphrey, "Forecasting cloud application workloads with cloudinsight for predictive resource management," *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 1848–1863, 2022.
[61] J. Han, Y. Hong, and J. Kim, "Refining microservices placement employing workload profiling over multiple kubernetes clusters," *IEEE Access*, vol. 8, pp. 192 543–192 556, 2020.
[62] S. Arora, A. Ksentini, and C. Bonnet, "Lightweight edge slice orchestration framework," in *ICC 2022 - IEEE International Conference on Communications*, 2022, pp. 865–870.
[63] N. F. V. N. R. . A. R. on the Enhancements of the NFV architecture towards Cloud-native and E. G. N.-I. . V. PaaS", 2019.
[64] Prometheus. (June 2024). [Online]. Available: https://prometheus.io
[65] E. Maps. (June 2024). [Online]. Available: https://app.electricitymaps.com/map
[66] N. G. E. C. I. API. (June 2024). [Online]. Available: https://carbonintensity.org.uk
[67] Submariner. [Online]. Available: https://submariner.io/
[68] Skupper. [Online]. Available: https://skupper.io/
[69] Multus. [Online]. Available: https://github.com/k8snetworkplumbingwg/multus-cni
[70] Calico. [Online]. Available: https://www.tigera.io/project-calico/

MOHAMED MEKKI Mohamed Mekki obtained his PhD from EURECOM, where he now works as a researcher. He specializes in the Cloud Edge Computing Continuum, focusing on utilizing containerization and WebAssembly technologies to enhance the automated management and deployment of applications, while also considering energy consumption and carbon footprint optimization. His work contributes to several European projects, including AC3, FLECON-6G, 6G-INTENSE and 5GDrones.

ADLEN KSENTINI Adlen Ksentini is a professor in the Communication Systems Department of EURECOM. He is leading the Network softwarization group activities related to Network softwarization, 5G/6G, and Edge Computing. Adlen Ksentini's research interests are Network Sofwerization and Network Cloudification, focusing on topics related to network virtualization, Software Defined Networking (SDN), and Edge Computing for 5G and 6G networks. He has been participating to several H2020 and Horizon Europe projects on 5G and beyond, such as 5G!Pagoda, 5GTransformer, 5G!Drones, MonB5G, ImagineB5G, 6GBricks, 6G-Intense, Sunrise-6G and AC3. He is the technical manager of 6G-Intense and AC3, on zero-touch management of 6G resources and applications, and Cloud Edge Continuum, respectively. He is interested in the system and architectural issues but also in algorithm problems related to those topics, using Markov Chains, Optimization algorithms, and Machine Learning (ML). Adlen Ksentini has given several tutorials in IEEE international conferences, IEEE Globecom 2015, IEEEE CCNC 2017/2018/2023, IEEE ICC 2017, IEEE/IFIP IM 2017, IEEE School 2019. Adlen Ksentini is a member of the OAI board of directors, where he is in charge of OAI 5G Core Network and O- RAN management (O1, E2) for OAI RAN activities.

MILOUD BAGAA Miloud Bagaa currently is a professor in the department of electrical and computer engineering of UQTR. He received the engineer's, master's, and Ph.D. degrees from the University of Science and Technology Houari Boumediene, Algiers, Algeria, in 2005, 2008, and 2014, respectively. From 2009 to 2015, he was a researcher with the Research Center on Scientific and Technical Information, Algiers. From 2015 to 2016, he was postdoctoral researcher with the Norwegian University of Science and Technology, Trondheim, Norway. From 2016 to 2019, he was a postdoctoral researcher at Aalto University. From 2019 to 2020, he was a senior researcher at Aalto University. Last but not least, he was a visiting researcher at Aalto University and a senior system cloud specialist at CSC from Oct. 2020 until Nov. 2022.