SN Computer Science (2024) 5:318
https://doi.org/10.1007/s42979-024-02630-w

ORIGINAL RESEARCH

Performance Optimization Across the Edge-Cloud Continuum: A Multi-agent Rollout Approach for Cloud-Native Application Workload Placement

Polyzois Soumplis[1,2] · Georgios Kontos[1,2] · Panagiotis Kokkinos[2,3] · Aristotelis Kretsis[1,2] · Sergio Barrachina-Muñoz[4] · Rasoul Nikbakht[4] · Jorge Baranda[4] · Miquel Payaró[4] · Josep Mangues-Bafalluy[4] · Emmanuel Varvarigos[4]

Received: 21 September 2023 / Accepted: 17 January 2024

Abstract

The advancements in virtualization technologies and distributed computing infrastructures have sparked the development of cloud-native applications. This is grounded in the breakdown of a monolithic application into smaller, loosely connected components, often referred to as microservices, enabling enhancements in the application's performance, flexibility, and resilience, along with better resource utilization. When optimizing the performance of cloud-native applications, specific demands arise in terms of application latency and communication delays between microservices that are not taken into consideration by generic orchestration algorithms. In this work, we propose mechanisms for automating the allocation of computing resources to optimize the service delivery of cloud-native applications over the edge-cloud continuum. We initially introduce the problem's Mixed Integer Linear Programming (MILP) formulation. Given the potentially overwhelming execution time for real-sized problems, we propose a greedy algorithm, which allocates resources sequentially in a best-fit manner. To further improve the performance, we introduce a multi-agent rollout mechanism that evaluates the immediate effect of decisions but also leverages the underlying greedy heuristic to simulate the decisions anticipated from other agents, encapsulating this in a Reinforcement Learning framework. This approach allows us to effectively manage the performance–execution time trade-off and enhance performance by controlling the exploration of the Rollout mechanism. This flexibility ensures that the system remains adaptive to varied scenarios, making the most of the available computational resources while still ensuring high-quality decisions.

Keywords Cloud native applications · Cloud edge continuum · Resource allocation · Multi-agent rollout · Reinforcement learning

This article is part of the topical collection "Recent Trends on Cloud Computing and Services Science" guest edited by Claus Pahl and Maarten van Steen.

✉ Polyzois Soumplis
soumplis@mail.ntua.gr

✉ Georgios Kontos
giorgoskontos@mail.ntua.gr

1 School of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece

2 Institute of Communication and Computer Systems, Athens, Greece

3 Department of Digital Systems, University of Peloponnese, Sparta, Greece

4 Services as Networks (SaS), Centre Tecnològic Telecomunicacions CTTC/CERCA), Catalunya, Spain

Introduction

Monolithic application design centers on a single, unified, and inseparable entity [1]. Historically, this paradigm was exceptionally efficient during times when applications encompassed a client-side user interface, a server-side logic, and an associated database. However, as the landscape of Information and Communication Technologies evolved [2–4]—with the advent of innovations like virtualization, optical networks, and the transitions to 5 G and 6 G—the complexity of the underlying application design increased. This created the constant need for refinements and optimizations to meet the ever-increasing Quality of Service (QoS) demands. In such a competitive and volatile environment,

Published online: 13 March 2024

the monolithic approach stands inadequate, leading to the emergence of the cloud-native application model.

In response to this evolving landscape, the principle of cloud-native application design has gained significant traction. Its cornerstone lies in the effective decomposition into microservices. These are discrete, loosely coupled components, each encapsulating its own logic, executing independently, and fulfilling a distinct function. Typically, microservices are executed within isolated virtual computing frameworks, such as containers [5]. These containers allocate the necessary computing, networking, and storage capacities directly from the host operating system. The cloud computing model seems to be a favorable candidate for enabling the scalable and dynamic operation of cloud-native applications. Through this, the microservices of an application can be deployed and scaled independently [6].

Today, a myriad of emerging applications, services, and digitized assets generate vast volumes of data at the network's edge, necessitating ultra-low processing delays. This encompasses innovative domains such as autonomous vehicles, smart cities, virtual/augmented reality, and biomedical care, which all harness a plethora of sensors and data-generating mechanisms. In these settings, relaying the generated data to distant cloud data centers can increase network congestion. Moreover, this approach cannot satisfy the stringent latency requirements of certain applications. In response, the paradigm of edge computing has emerged [7]. Edge computing decentralizes resources, situating them proximate to the data origins. This ensures immediate access to computing, networking, and storage capabilities. While these edge units might not match the capacity of comprehensive cloud infrastructures, they significantly augment the performance of cloud-native applications by handling time-sensitive tasks. Hence, cloud resources need to be utilized in combination with the edge, e.g., serving latency-tolerant workloads, creating a powerful edge-cloud continuum. Software-wise, lightweight containers are the ideal technology to enable the seamless execution of cloud-native applications, or parts of them, on the edge [8], especially when compared to other virtualization approaches like virtual machines.

The proliferation of applications and their requirements necessitates the development of innovative mechanisms. These mechanisms must consider application-related characteristics, such as resource availability and latency when placing microservices. They should also account for the broader infrastructure's available resources. The challenge to optimize resource utilization across both edge and cloud infrastructures is a complex task. From a resource allocation standpoint, this involves operations like the placement of tasks, load balancing, and data management distributed across diverse computing landscapes. These challenges intensify given the volatile nature of multi-technology environments in edge and cloud computing and the unpredictable workloads nature, fluctuating resource availability, and potential infrastructure or application failures. The use of a global scheduler to dictate the operation of the different domains in a centralized manner is not applicable due to this high complexity which results in unaffordable slow execution time. In this direction, multi-agent algorithms are particularly well-suited as they can advantageously provision resources to applications' workload in a distributed manner speeding up the process, while also incorporating real-time monitoring and historical telemetry data, and even pricing information.

The present work focuses on developing a multi-agent rollout mechanism for allocating the available computing and storage resources in the various layers of an edge-cloud infrastructure to fulfill the diverse computing, storage, and latency constraints of the incoming workloads from cloud-native applications. The proposed mechanisms, offline in nature, can seamlessly leverage infrastructure telemetry mechanisms to enable a periodic reevaluation and adjustment of resource allocation strategies in anticipation of the expected workloads. By integrating this dynamic data, our algorithms can adapt to changing application needs in a reinforcement learning manner to ensure efficient resource utilization even in evolving operational contexts. We aim to jointly optimize a weighted combination of the average delay (per application) and the average cost of service while ensuring that the delay between dependent microservices and the available resources on the infrastructure nodes meets the requirements specified by the applications.

We first provide the Mixed Integer Linear Programming (MILP) model of the problem that is able to track the optimal solution. Recognizing the potentially prohibitive execution time for real-sized scenarios, we then introduce a greedy best-fit heuristic that achieves a considerably lower execution time. A multi-agent rollout mechanism is also proposed to exploit further the trade-off between execution time and performance. This reinforcement learning mechanism assesses the cost of immediate and future decisions during the resource allocation process in order to decide for a given allocation. The estimation of the future cost is based on the greedy heuristic, and it simulates the decisions that will be made by the agents. Our evaluation, conducted through extensive simulations leveraging publically available traces of cloud providers, across a variety of scenarios, showcases the effectiveness of the proposed solutions in delivering timely decisions, adeptly balancing execution time and performance.

The remainder of this work is organized as follows: In Sect. "Related Work", we report on the related work. In Sect. "Resource Allocation Mechanisms", we present the considered edge-cloud infrastructure model and the cloud-native applications workload. Next, we present the MILP problem formulation, the heuristic, and the multi-agent

rollout algorithm. In Sect. "Simulation Experiments", we present the simulation results and finally, in Sect. "Conclusions", we conclude our work.

Related Work

The resource allocation problem in virtualized environments is a multi-dimensional research area that has attracted the interest of the research community. The modeling of the problem among the different works varies according to the considered topology and the adopted technologies, while the proposed solutions employ techniques from the wider realms of mathematics and computer science.

In [9], the study addresses the placement of Virtual Machines (VMs) on physical systems in cloud data centers for IoT big-data analytics. Using a graph model with nodes as VMs and links representing network communication, the goal is to evenly distribute link utilization, thus averting congestion. A greedy heuristic is employed to place closely interacting VMs on the same system, reducing communication costs. Meanwhile, [10] introduces the VNFPRA problem, which aims for optimal Virtual Network Function (VNF) placement in SDN-NFV-enabled multi-access edge computing (MEC) nodes, targeting minimal deployment and resource costs. The infrastructure is depicted using a weighted graph with nodes representing MEC capacities and links signifying network capacities. Formulated as a MILP problem, the total service cost encompasses placement, resource usage, and link costs, with a genetic-based heuristic proposed for efficiency. Authors of [11] employ a Gaussian Process Regression-based algorithm to predict traffic, focusing on timely request servicing. A hierarchical infrastructure is leveraged to prioritize time-sensitive demands. Lastly, the work in [12] tackles the problem of resource allocation for IoT machine learning applications over the edge-cloud continuum. The authors consider the continuous movement of sensors' data to computing machines for the efficient training of machine learning models. A balance between cost, execution and communication delay, and accuracy is explored. A Mixed Integer Linear Programming formulation is presented to optimally solve the problem, while a heuristic followed by a simulated annealing is developed to provide a faster, albeit sub-optimal, solution.

In [13], a decentralized approach is emphasized for selecting an appropriate edge-fog node to orchestrate IoT workloads. Their heuristic evaluates both application requirements and fog node conditions to identify the most suitable Fog Orchestrator Node (FON). [14] tackles the mapping challenge between end devices and edge gateways, aiming to reduce communication costs and maintain load balance. They introduce a particle swarm optimization metaheuristic where initial solutions adopt exploration or exploitation tactics. The method's efficacy is validated through simulations. In [15], a QoE-aware placement policy for IoT applications in fog environments is proposed. Employing two fuzzy logic models, they evaluate application expectations and fog node performance and feed a linear optimization mechanism to perform the placement.

Reinforcement learning has been pivotal in resource allocation within edge-cloud domains. In [16], a state-action-reward-state-action-based deep reinforcement learning method addresses task off-loading in MEC. By modeling user requests as sub-tasks and considering execution options (nearest edge, adjacent edge, or central cloud), authors aim for reduced service delays and energy use. Authors in [4] confront a microservice coordination mechanism in mobile edge computing. With mobile users like autonomous vehicles offloading computation, the challenge is to mitigate both delay and migration costs. An optimal offline algorithm is presented, complemented by a Q-learning approach that offers near-real-time, near-optimal solutions. Further, [17] details a deep reinforcement learning strategy for microservice deployment across heterogeneous edge-cloud terrains. Viewing microservices as ordered service chains, they aim to optimize the Average Waiting Time (AWT) of these services. Finally, Yang et al. [18] approach the problem of partial-offloading of end-user applications by employing a tandem-queue system comprising edge and cloud processing. Request arrivals, service times and computation resources are considered to be stochastic parameters. A DRL agent is exploited to make the decisions of the computational speed of each queue by appropriately allocating computational resources, with an objective of avoiding resource wastage while minimizing the probability of delay-constraints violation.

In distributed infrastructures, timely communication between microservices is paramount to prevent computational inconsistencies. Interactions, either sequential or parallel, are pivotal. To address this, Directed Acyclic Graph (DAG)-based methods have been proposed for resource allocation. In [19], the authors target the minimization of resources and virtual machine instance hours for DAG-based and deadline-constrained applications in the cloud. They developed the MSMD (minimal slack time and minimal distance) algorithm, aiming to optimize the number of VM instances while meeting application deadlines. The operation of this algorithm is further supported by a heuristic approach that shuts down idle VMs. [20] emphasizes the necessity of balancing communication processes with computing tasks in cloud resource allocation. They introduce the CA-DAG model, offering a scheduling technique that distinctly allocates resources for computing and communication requirements. Meanwhile, authors in [21] propose a cost-optimized DAG scheduling algorithm in IaaS cloud

platforms, as a response to the emergence of cost-centric public cloud services.

In this work, we present the first, to the best of our knowledge, application of the multi-agent rollout technique in such a context. This optimization approach, rooted in the principles of Approximate Dynamic Programming and Reinforcement Learning, is used to consider both immediate and future decision impacts. The cost of immediate decisions is directly assessed through the cost function, while the cost of future decisions is approximated by the greedy heuristic algorithm. This heuristic aids in estimating the cost associated with resource assignments tied to subsequent agents, thereby equipping the model with enriched information for more judicious decision-making. Different parameters are used to decide the effect of future actions and the exploration of the immediate actions during the resource allocation process.

Resource Allocation Mechanisms

The Edge-Cloud Infrastructure

We consider a hierarchical edge-cloud infrastructure, with multiple layers of edge resources (e.g., on-device, near-edge, far-edge) to serve the incoming cloud-native workload. We assume that the edge layers consist of machines with relatively limited resources, such as Raspberry Pi's, NVIDIA Jetson, servers, mini-data centers, etc., while the cloud layer has practically unlimited resources.

The hierarchical edge-cloud infrastructure is denoted by an Undirected Weighted Graph $G = (V, E)$. The nodes $v \in V$ of the graph correspond to the locations where computing resources are available. Each location is described by a tuple of the available resources $\tau_v = [c_v, r_v, o_v, n_v]$, where c_v is node's v CPU capacity measured in CPU units, r_v is the node's RAM capacity measured in RAM units, o_v is the node's operating cost, and n_v is the node's networking cost coefficient. In our case, operational cost relates to the expenses made for purchasing, deploying, and operating the respective computing/storage systems. As providers achieve economies of scale the operational cost is small for the cloud layer, and gradually increases as we move to the lower edge layers, which are characterized by their geographically dispersed placement, a limited number of resources, and a small number of users. Networking cost coefficient n_v results from the usage of any link from the nodes where data are generated, to the node(s) v where computing operations take place and is multiplied by the ingress data to deduce the actual networking cost of service. The coefficient is minimal for the near-edge nodes, where links are shorter in distance and cheaper to install, while it gradually increases up to the massive links connecting the cloud nodes. Generally, data is generated at the lower levels of the infrastructure that can be either equipped with computing resources (local processing) or not. As they are typically located on the near edge, the delay is small for transferring the data to a subset of near edge nodes as they are located closer to the data source, given their plurality and thus higher geographical density, while it increases for the higher layer nodes (far edge, cloud). Finally, each link $e \in E$ between two nodes v and v' is characterized by a weight $l_{v,v'}$, representing the communication (propagation) delay of nodes v and v'.

The workload under consideration consists of a set A of cloud-native applications. Each application $a \in A$ is described by an Undirected Weighted Graph $G^a = (V^a, E^a)$, with the nodes V^a corresponding to the microservices that make up the application and the arcs E^a representing the inter-dependencies (communication requirements) among them. Each cloud-native application has a source node $\pi_a \in V$ and each microservice $i = 1, \ldots, |I^a|$ of application a, has specific resource requirements described by the tuple $[\epsilon_{a,i}, \rho_{a,i}, s_{a,i}]$, where $\epsilon_{a,i}$ is the microservice's CPU demand, $\rho_{a,i}$ is its memory demand, and $s_{a,i}$ is the size of the data to be transferred. Furthermore, each arc $e \in E^a$ between two microservices $i, i' \in V^a$ is characterized by a weight $\lambda_{a_{i,i'}}$ that represents the maximum acceptable delay between the corresponding service nodes v, v'. This is a measure of the intensity of the dependency between these two microservices, in the sense that highly dependent microservices should be served by the same or geographically approximate nodes to reduce communication costs and guarantee the application's efficiency with in-time calculations. In Table 1 we summarize all the notations used throughout this section for clarity and ease of understanding.

Mixed Integer Linear Programming (MILP) Formulation

In this subsection, we present the mathematical model of the resource allocation problem encountered by cloud-native applications within an integrated edge-cloud architecture, initially introduced in [22]. The core aim is to optimize the weighted sum of the average operational and networking costs and the maximum delay for data communication for each application, under the constraints set by computational and networking constraints, application-specific needs, and node resources.

Objective Function:

$$\min \left\{ w \sum_{a=1}^{A} \sigma_a + (1 - w) \sum_{a=1}^{A} \theta_a \right\} \tag{1}$$

Subject to the following constraints:

Table 1 Summary of notations

Notation	Interpretation
G	Undirected Weighted Graph representing the infrastructure
G^a	Graph representing microservices and dependencies of application $a \in A$
V	Set of infrastructure nodes
τ_v	The tuple representing the computing characteristics and associated monetary cost of node $v \in V$
c_v	CPU capacity of node $v \in V$
r_v	RAM capacity of node $v \in V$
o_v	Operating cost of node $v \in V$
n_v	Networking cost coefficient of node $v \in V$
$l_{v,v'}$	Communication delay between nodes $v, v' \in V$
A	Set of cloud-native applications
p_a	Source node of application $a \in A$
I_a	Set of microservices of application $a \in A$
$\epsilon_{a,i}$	CPU demand of microservice $i \in I_a$
$\rho_{a,i}$	RAM demand of microservice $i \in I_a$
$s_{a,i}$	Input size for microservice $i \in I_a$
$\lambda_{a_{i,i'}}$	Maximum acceptable delay between the service nodes of microservices ι, ι'
$x_{v,a,i}$	Binary variable for allocation of microservice $i \in I_a$ to node $v \in V$
σ_a	Total monetary cost for serving application $a \in A$
θ_a	Communication delay for application $a \in A$
w	Weighting coefficient controlling the objective
$\mathcal{X}$	State space in the multi-Agent Rollout mechanism
$\mathcal{T}$	Available resources of infrastructure nodes in state space
$\mathcal{W}$	Remaining microservices pending service in state space
$\mathcal{Q}$	Allocation of nodes for microservices within the current application
$\mathcal{U}$	Action space comprising all feasible actions from a given state
R	Reward function in the Rollout mechanism
$g_k(x_k, u_k)$	Scalar value reward function for transition due to action u_k
$J_k^*(x_k)$	Cost function representing the optimal cost action for state k
γ	Discount factor in the Rollout mechanism

C.1. Placement of the microservices to nodes. For each application $a = 1, \ldots, A$ and for each microservice $i = 1, \ldots, I_a$

$$\sum_{v=1}^{V} x_{v,a,i} = 1 \tag{2}$$

C.2. Respect of the relative latency between the applications' microservices. For each application $a = 1, \ldots, A$, and each pair of microservices of application $a, i, i^{'} = 1, \ldots, I_a$,

$$l_{v,v'} x_{v,a,i} + l_{v,v'} x_{v',a,i'} \leq \lambda_{a,i,i'} + l_{v,v'} \tag{3}$$

C.3. The allocated CPU units of the served applications cannot surpass the number of available CPU units at each node. For each node $v = 1, \ldots, V$,

$$\sum_{a=1}^{A} \sum_{i=1}^{I_a} \epsilon_{a,i} x_{v,a,i} \leq c_v \tag{4}$$

C.4. The allocated memory units of the served applications cannot surpass the number of available memory units at each node. For each node $v = 1, \ldots, V$,

$$\sum_{a=1}^{A} \sum_{i=1}^{I_a} \rho_{a,i} x_{v,a,i} \leq r_v \tag{5}$$

C.5. Total monetary application cost σ_a calculation. For each application $a = 1, \ldots, A$,

$$\sigma_a = \sum_{v=1}^{V} \sum_{i=1}^{I_a} (o_v + n_v s_{a,i}) x_{v,a,i} \tag{6}$$

C.6. Maximum per application latency (propagation) calculation. For each node $v = 1, \ldots, V$, for each cloud-native application $a = 1, \ldots, A$, and each of its microservices $i = 1, \ldots, I_a$,

$$\theta_a \geq x_{v,a,i} l_{\pi_a,v} \tag{7}$$

The objective function (Eq. 1) is the weighted sum of the maximum delay and cost per application assignment, where $w = 0$ considers purely the delay minimization problem and thus the microservices of the applications are preferably placed in the edge (horizontal scaling), $w = 1$ deals with the cost minimization problem and thus the vertical scaling of applications, and any intermediate value of w considers both of the aforementioned parameters, with the different contribution in the calculation of the total objective value. The first constraint (Eq. 2) is used to ensure that every microservice is assigned to exactly one node. The second constraint (Eq. 3) enforces that the allocation of resources among interacting applications of a microservice, is performed with respect to their latency constraint. Constraints (3) and (4) ensure that microservices running in a node do not use more than the available resources, while constraints (5) and (6) calculate the monetary cost and the maximum latency of each application respectively. Note that our considered formulation supports general workloads (not strictly cloud-native applications) that can take the form of an application with a single microservice.

The total number of variables required by the MILP formulation, given an infrastructure with $|V|$ nodes that serve $|A|$ cloud-native applications, each comprised of $|I|$ microservices, is $[|V| \cdot |A| \cdot |I| + 2 \cdot |A|]$. The formulation entails $|A| \cdot |I|$ equality constraints for constraint 1 (Eq. 2 C.1.) and an additional $|A|$ for constraint 5 (Eq. 6). The inequality constraints include $|A| \cdot |I|^2 \cdot |V|^2$ for constraint 2 (Eq. 3), $|V|$ each for constraints 3 and 4 (Eqs. 4 and 5 respectively), and $|V| \cdot |A| \cdot |I|$ for constraint 6 (Eq. 7).

Greedy Best-Fit Resource Allocation Algorithm

The considered problem belongs to the NP-hard class of problems [23] the presented MILP is computationally intensive with prohibitively large execution time even for small-sized problems. For this reason, we developed suboptimal mechanisms. First, we present a greedy heuristic that performs the provisioning of resources of the microservices sequentially in a best-fit manner. Next, we present the multi-Agent Rollout mechanism, a meta-heuristic that leverages the developed greedy heuristic in a reinforcement learning manner to provide an improved solution through an iterative process.

The considered greedy heuristic [22] seeks to find a satisfactory solution by serving the application demands in a best-fit manner. It takes as input the infrastructure graph $G = (V, E)$ along with all the applications' demands and its microservices described by graph $G^a = (V^a, E^a)$ for application a, $\forall a = 1, \ldots, A$ and serves the applications sequentially, one by one. After selecting an application, the first microservice of the application is selected and the candidate infrastructure nodes with enough resources are calculated in order to accommodate it. These nodes are ranked based on the objective function considering the cost and the latency introduced by the assignment of the microservice $i = 1, \ldots, I_a$ to each node. The best node $v \in V$ is selected and the processing and memory resources demanded by the microservice are reserved. If the application consists of more than one microservice, the next microservice is selected. The same process is followed for the following microservice with the addition of the relative latency constraint between the communicating microservices.

Hence, given the first microservice location, the mechanism selects the nodes $v' \in V$ that exhibit communication latency smaller than the limit set by the examined microservice, $l_{v,v'} \leq \lambda_{\mu_{a,1},\mu_{a,2}}$. If more than one node is found, it places the second microservice in the best one (it could be the same node as the first microservice). The same process is repeated until the I_a-th microservice of the application is served. If it is not possible to find a node to host an application's microservice, the procedure is re-initiated for the same application considering the second-best node for the first microservice and so on. When a solution is found, the utilization of the resources is updated and the application is marked as served. The above process is repeated for all applications, returning the final assignment and the value of the objective function (Eq. 1). Note that from the description of the procedure, it may be the case where the selection of the first node can make the execution of an application impossible due to the latency constraint among the microservices of the application. Although this may happen for the edge resources which are characterized by a limited capacity of resources, this does not stand for the abundant cloud resources, which are able to execute the application demands at the price of increased propagation latency. The complexity of this approach is polynomial with a worst-case execution time of $O(|A| \cdot |I_a| \cdot |V|^2)$, assuming that all the nodes $|V|$ are candidate locations to serve the first microservice of each application, and all possible re-allocations take place. A typical iteration of this algorithm is presented in the flowchart of Fig. 1.

Multi-Agent Rollout Mechanism

To further improve the performance of the aforementioned greedy heuristic, we considered a multi-agent rollout mechanism, inspired by approximate dynamic programming [24], a well-known reinforcement learning technique. The mechanism comprises multiple agents, one per microservice, which operate sequentially, trying to meet the assigned microservice's requirements. Each agent allocates resources to serve the assigned microservice with respect to the constraints set by the cloud-native application and the infrastructure and communicates its decision to the remaining agents, which are therefore accurately informed about the

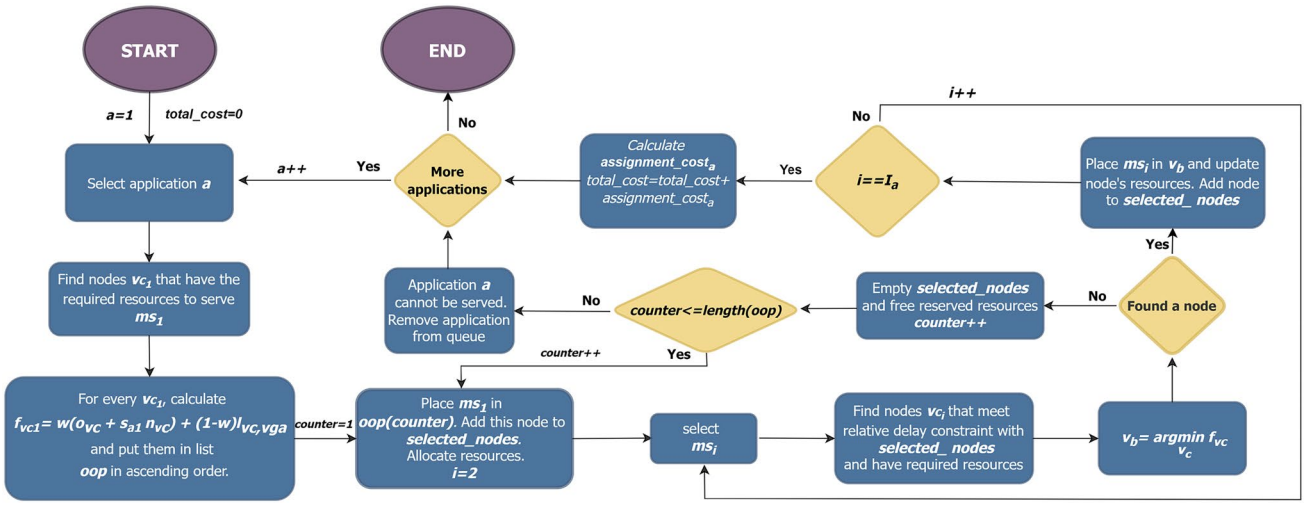

Fig. 1 The flowchart of the best-fit heuristic

availability of the infrastructure resources, the assignment of the microservices that make up the application and the remaining microservices that need to be served.

This is reflected in the state space $\mathcal{X}$ of the system which is defined as $\mathcal{X} = (\mathcal{T}, \mathcal{W}, \mathcal{A})$, where $\mathcal{T}$ represents the available resources of the infrastructure nodes, captured as a set of tuples $\mathcal{T} = \bigcup_{v \in V} \tau_v$ with each τ_v specifying the utilization on node v in CPU and RAM; $\mathcal{W}$ represents the remaining microservices that are pending service; and $\mathcal{Q}$ details the allocation of nodes for the microservices within the current application, where each allocation q_i signifies the node assignment for microservice i.

The agent receives as input the partial solution of the problem as described by the current state and examines the different actions that can be made to allocate resources for the assigned microservice. The action space $\mathcal{U}$ comprises all feasible actions from any given state. Specifically, for state k, an action $u_k \in \mathcal{U}$ corresponds to the assignment of a microservice to an infrastructure node V. An action is deemed feasible when it meets certain requirements: it must adhere to the capacity constraint, meaning a node v's available resources must be sufficient for the task it is allocated, and the latency constraint, which ensures that the latency between nodes managing intercommunicating tasks does not surpass the established acceptable limits.

When an action is applied, the system transitions to the next state while a reward is obtained. The reward function R is determined by the reward function $g_k(x_k, u_k)$, which assigns a scalar value to the transition from state x_k to x_{k+1} due to action u_k and is the cost obtained by the objective function of the considered optimization problem. To approximate the future cost to serve the remaining microservices, the presented greedy best-fit heuristic is used as a base policy to account for the remaining microservices. Among all possible actions, the one ($J_k^*(x_k)$) that minimizes the summation of the immediate cost x_k that marks the transition to state x_{k+1}, and future cost approximation from state x_{k+1}, $J_{k+1}^*(x_k)$, is selected (eq. 8).

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} \left[g_k(x_k, u_k) + \gamma * J_{k+1}^*\left(x_k, u_k\right) \right] \tag{8}$$

In other words, from all the actions u_k for state k, the option with the least cost $J_k^*(x_k)$ is selected (Fig. 2). The node utilization serving the microservice is updated, marking until all application demands A are addressed. Finally, the assignment of microservices to nodes and the weighted cost of the assignment are returned.

As the action space for a given state can be huge the possible actions are ordered based on the immediate cost from the lowest to the greatest, with a number of actions u_k (up to a maximum of β) being computed, where u_k represents the possible placements for the current microservice containing all the nodes that respect the resource allocation constraints at the state x_k. If more than β placements exist, they are limited to the first β. This parameter becomes particularly useful when the exact methods are too slow and/or when solutions provided by heuristics are inefficient. The depth window, β, optimizes execution by regulating the count of distinct placements assessed for a given microservice during the immediate cost assessment. Despite potentially affecting the algorithm's efficacy, it proves valuable when multiple placements exist for a given microservice in controlling the execution time of the Rollout mechanism.

Another critical parameter is the use of a discount factor γ which adjusts the influence of the heuristic's future cost approximation on the current action selection. The incorporation of γ, a key aspect of the standard infinite horizon discounted MDP, becomes especially beneficial when the

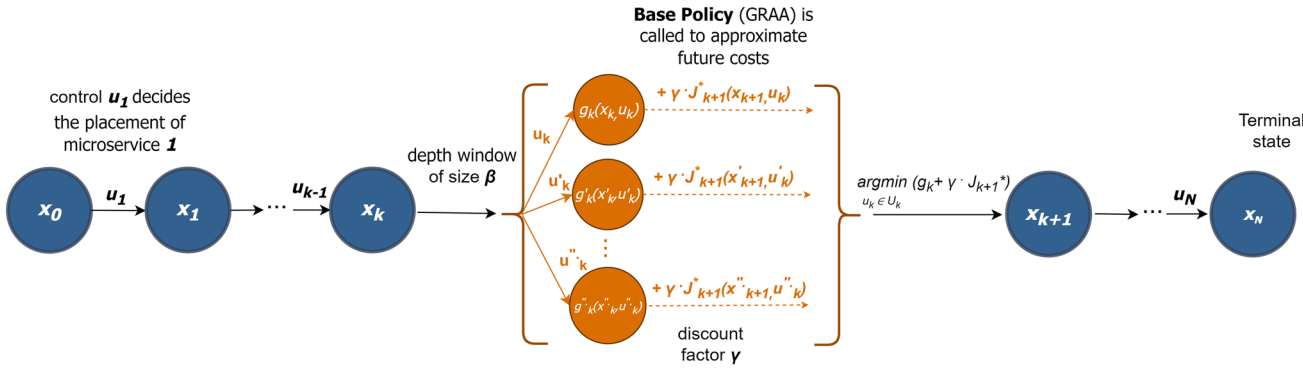

Fig. 2 The different multi-Agent Rollout options for serving the i-th microservice of application k

performance of the base policy does not meet expectations, allowing for an adjusted approach that can yield better immediate-term outcomes without completely disregarding future implications. The practical significance of the discount factor in our model lies in scenarios where telemetry data may not provide accurate estimations, reinforcing the necessity of a mechanism that can adapt to uncertainties in the operational environment. By judiciously adjusting γ, our system maintains a proactive stance in resource allocation, thus ensuring efficiency under different conditions.

A significant advantage of this method is that exploits the trade off between the complexity of the action space with new states, making computational demands proportional to the microservice and the infrastructure's node count. Consider an application with I_a microservices. Each microservice's potential placement in any of the V infrastructure nodes can lead to $|V|^{|I_a|}$ different choices for the application's collective decision. Dividing application a resource allocation into $|I_a|$ sequential decisions and applying agent-by-agent rather than all simultaneously reduces the different choices to $|V| \cdot |I_a|$.

Simulation Experiments

In this section, we delve into the simulation experiments conducted to evaluate the performance of the proposed mechanisms. These experiments not only offer a thorough evaluation of the mechanisms' performance but also provide deep insights into the problem's complexities. The proposed mechanisms were implemented in MATLAB and the simulation experiments were conducted on a 6-core 2.6 GHz Intel Core i7 PC with 12 GB of RAM.

Simulation Setup

We assume an infrastructure that spans over the edge-cloud continuum and is organized into three distinct layers of computing resources, each characterized by its unique capabilities. These layers range from powerful cloud servers, offering substantial processing power, to far and near-edge resources, which provide localized, rapid-response capabilities with lower computing capacity compared to the cloud ones.

To cater to different experimental scenarios, we examined two distinct edge-cloud topologies. The first topology, namely the "basic topology", comprises 20 near-edge, 5 far-edge and 2 cloud nodes. This limited configuration allows the evaluation of the proposed sub-optimal mechanisms against the optimal performance given by the MILP mechanism, which can be efficiently executed in such small-scale problem instances. We also assumed, an upscaled configuration, namely the "extended topology" that encompasses 150 near-edge nodes, 30 far-edge nodes and 7 cloud nodes. This setup allows a better understanding of the problem's dynamics while showcasing the scalability of our proposed sub-optimal mechanisms.

We modeled cloud resources, comprising a cluster of 400 to 600 CPU cores per node as profiled in [25] and memory size in the interval [600, 800] GBs. The far-edge layer was assumed to include nodes with [40, 80] CPU cores and [50, 100] GB RAM. Finally, near-edge computing nodes were assumed to align with the specification of devices indicative in such environments, hence we considered near-edge nodes equipped with [1,8] cores and GB of RAM respectively. In our model, the monetary cost of service is normalized within the [1,10] cost units (c.u.) interval, while the variance in costs across different computing layers—cloud, near-edge, and far-edge—is derived from the differences outlined in [26]. Near-edge resources incur the highest costs, reflecting their geographical scarcity and confined availability. Conversely, cloud nodes offer the most cost-efficient service due to their centralized nature, while far-edge resources fall in between. These specifications cost distributions across the different layers are detailed in Table 2. The networking cost coefficient values are based on [27], effectively considering

Table 2 Node characteristics on different topology layers

Layer	Nodes	CPU (cores)	Mem. (GB)	Mon. Cost (c.u.)	Net. cost (l.u.)
Near Edge	50	[1, 4]	[1, 8]	[6, 8]	[0.1,0.2]
Far Edge	10	[40, 80]	[50, 100]	[3, 4]	[0.4,0.6]
Cloud	5	[400,600]	[600,800]	[1, 2]	[1,1.2]

Table 3 Communication delays between layers (values in units)

	Data source	Near edge	Far edge	Cloud
Near edge	[1, 2]	[0.5, 2]	[2, 4]	[5, 6]
Far Edge	[3, 4]		[2, 3]	[3, 4]
Cloud	[7, 8]			[4, 5]

the cheaper networking cost of edge networks and are presented in latency units (l.u.).

Finally, the communication delays among the different infrastructure nodes are presented in (Table 3). The considered propagation delays are aligned with the inherent characteristics of each layer: Near-edge nodes are situated in close proximity to the data source, hence incurring the lowest delay. Conversely, cloud nodes, located in distant areas, experience prolonged delays, while the far-edge layer stands amidst the near-edge and cloud, providing moderate delays [28].

The workload for our study is based on cloud-native applications, composed of various microservices. These microservices' specific CPU and RAM demands are synthesized by leveraging publicly available, real-world datasets, as elaborated in [29]. This dataset includes traces of both online services and batch jobs. For the design of our workload, we specifically focused on the resource consumption patterns of online services, which are engineered for real-time user interaction. These services typically require consistent and predictable resource allocation to achieve performance benchmarks, such as low latency.

Optimality Performance Evaluation of Heuristic and Multi-Agent Rollout Mechanism

In Table 4, we initially evaluate the performance of the multi-agent rollout and the greedy heuristic against the optimal MILP solution in terms of execution time and objective cost. These evaluations are based on workloads that consist of a ranging number of cloud-native applications from 50 to 300. We set the depth size parameters at $\beta = 90\%$ and $\gamma = 0.8$ and utilize the basic topology. The primary focus of our objective function is to optimize latency, with a minimal inclusion of cost. If we were to optimize solely for monetary cost, microservices would predominantly utilize cloud resources due to their high availability, resulting in most placements being in the cloud.

In assessing the performance of the proposed mechanisms, the best-fit heuristic achieved the worst performance, with a gap of up to 10% from the optimal solution, whereas the multi-Agent rollout managed to generate solutions within 4% of the optimal in all cases. As for the execution time, the greedy heuristic exposed the lowest execution time in the order of milliseconds even for higher workloads, while rollout's execution time growth is polynomial with the workload increment. Finally, the MILP solver had exponentially increasing execution times, and for the workload sizes of 250 and 300 application demands, it finished its operation within the time limit that was set, while for the largest workload did not manage to produce a feasible solution during this period.

Evaluating the Performance Based on the Different Objectives

In this sub-section, we explore how varying objectives influence system performance, as illustrated in our allocative efficiency analysis presented in Fig. 3. For our experiments

Table 4 The total cost and the execution time for $w = 0.01$ for the different mechanisms

Application demands	MILP		Multi-Agent Rollout		Heuristic	
	Obj. value	Exec. time	Obj.Value	Exec.time	Obj.Value	Exec.Time
50	52.14	17.41	55.12	8.33	57.14	0.06
100	107.13	306.35	109.81	45.67	116.54	0.18
150	231.71	3561.76	238.15	121.33	254.77	0.33
200	414.51	6513.13	433.68	245.63	451.09	0.44
250	667.04	10000	691.51	357.16	714.44	0.69
300	-	10000	991.64	411.11	1034.88	0.91

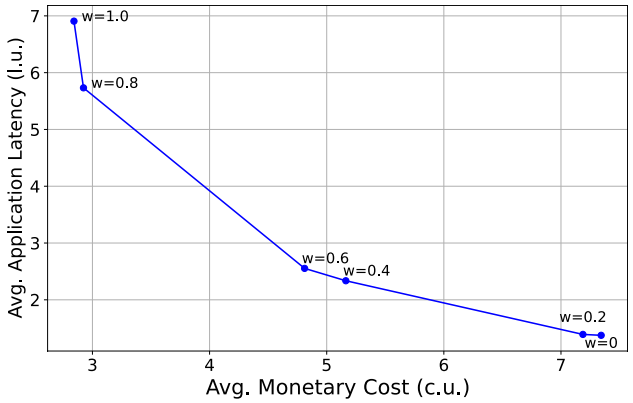

Fig. 3 The Pareto efficiency chart

we considered the extended topology and a workload demand that consists of 1000 cloud-native applications. As expected, the lowest cost is achieved when cloud and far-edge resources are highly utilized and thus the propagation latency increases as cloud resources are located in a few distant locations to which the data are transferred. When the single optimization criterion is the minimization of latency, the propagation delay is minimized by approximately 80% compared to the previous case, while the monetary cost is increased by almost 160%.

Next, we examined the utilization of edge and cloud resources for the different weighting coefficients w (Fig. 4) over the extended topology. Edge resources are utilized more in small weight values, as the objective is delay minimization and edge layers consist of nodes in geographic proximity to the data source. In this case, the microservices of an application expand over the resources of the edge layer. On the other hand, far-edge and cloud resources are heavily utilized in high w values, as the objective is approaching the monetary cost minimization, thus the "cheap" and powerful cloud nodes are preferred. For intermediate values of w, applications' microservices are allocated mostly across the far-edge layer, which strikes a fair balance between delay and cost. This showcases the importance of edge resources in the minimization of the latency of the applications for time-critical operations.

Figure 5 presents the contribution of networking and operational costs for the different weighting coefficient values. When the objective function targets the minimization of the monetary cost, the cloud resources are preferred with the networking costs contributing approximately 49% to the total cost, as the processing cost is low while the networking cost increases for transferring the application data to the cloud. On the other hand, when the objective is the minimization of latency and edge resources are utilized, the processing cost of the edge resources is the main factor of the total monetary cost, with the networking cost corresponding to a mere 5% of the total cost.

Evaluating the Influence of Depth Window Size and Discount Factor on the Performance of the Multi-Agent Rollout

While the multi-agent rollout mechanism achieves performance close to the optimal, its computational complexity can be significantly higher for larger configurations. In the next set of experiments, we seek to explore the dynamics of the depth window size β and future cost approximation γ when dealing with 1000 application demands and the extended topology. As the proposed heuristic algorithm exhibits linear complexity with regards to the workload size, albeit quadratic with the infrastructure, the multi-agent Rollout mechanism invokes the heuristic each time a new placement for a microservice (function) is evaluated, resulting in thousands or even tens of thousands of invocations in a

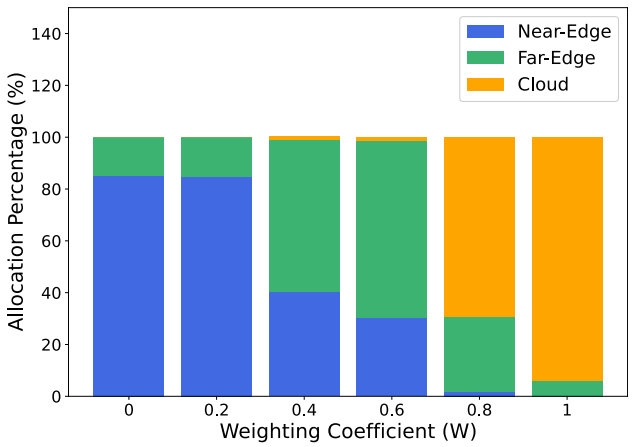

Fig. 4 The The number of microservices served at the near/far edge and the cloud layer

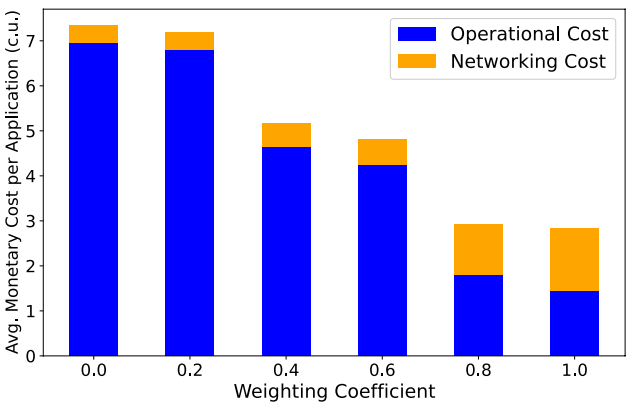

Fig. 5 The operational and networking cost for the different objective co-efficients

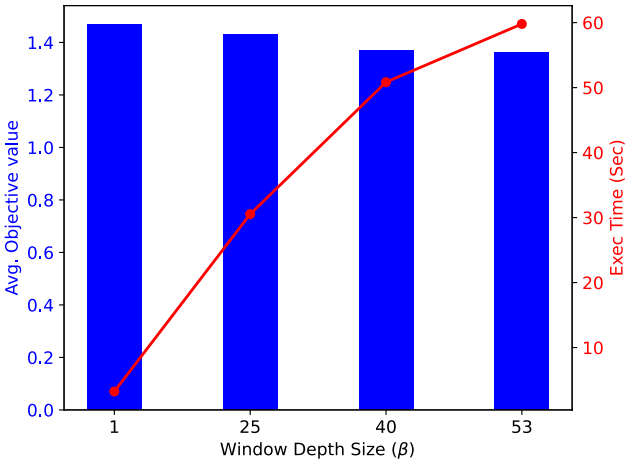

Fig. 6 The objective cost and the execution time for different sizes of depth window size β

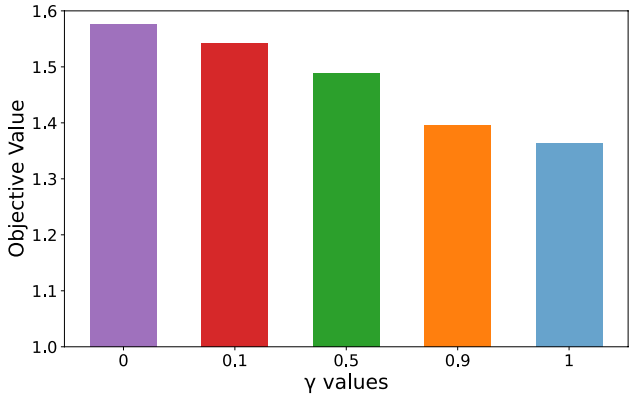

Fig. 7 The evolution of the objective value through time for different γ values)

single run. Execution time and scalability are Rollout's fundamental caveats. Properly fine-tuning Rollout's parameters β and γ can strike a balance between optimality and execution time depending on the characteristics of the examined workload. This flexibility in adjusting the parameters β and γ is particularly beneficial in real-case scenarios where workload characteristics are dynamic, allowing for tailored tuning to optimize performance under varying conditions.

In Fig. 6, we present the effect of the depth window size β on the performance of the algorithm and the average execution time for $\gamma = 0.8$. We observed that as the depth window size β increases, there is a noticeable impact on the algorithm's performance. Specifically, with a smaller β, the algorithm tends to converge faster and settles for an immediate action minimum as in the case of the greedy best-fit heuristic. On the other hand, a larger β allows for a more thorough exploration of the solution space but at the cost of computational efficiency. As expected, the average execution time rises with an increase in β as the calls to the heuristic for the future cost approximation increase. The results, however, also highlight the trade-offs involved: while higher values of β lead to better solutions, they can increase the execution time. However, by appropriately configuring parameter β, it is possible to achieve a balance between solution quality and computational efficiency. This balance is crucial for scenarios where timely decision-making is imperative.

In Fig. 7, we present the performance of the multi-agent Rollout mechanism with a depth window size of $\beta = 0.8$, focusing on the role of the future cost approximation parameter, γ. A higher γ places an emphasis on incorporating future cost approximations into current placement decisions for microservices, while a lower γ leads to a myopic view that favors the direct cost reduction. At $\gamma = 0$, the mechanism completely disregards future costs, aligning closely with a best-fit greedy heuristic. The difference in objective cost between the two extreme values of γ is almost 15% showcasing the need for accurate telemetry mechanisms. The efficacy of γ becomes particularly crucial when coupled with a telemetry mechanism capable of providing information regarding the future demands, enhancing performance by enabling the multi-agent rollout to fully leverage the provided data.

Conclusion

In this work, we tackled the challenge of resource allocation within multi-layered edge-cloud architectures, focusing on the optimal delivery of cloud-native applications. We incorporated several critical parameters, such as delay constraints arising from microservice communication and the availability of computing resources for serving cloud-native applications. Our proposed solution, the multi-agent rollout mechanism, effectively orchestrates resource allocation for various microservices of cloud-native applications, modeling the respective problem in the context of Reinforcement Learning. We employed a greedy best-fit heuristic for future cost approximation, capitalizing on its rapid execution and reliable estimations. Nonetheless, our mechanism retains its versatility, allowing the integration of alternative algorithms for future cost prediction. Furthermore, we introduced parameters that control the exploration intensity of the multi-agent rollout and manage execution time by restricting the solution space for each microservice. Our simulation results underscore the mechanism's adeptness at balancing delay against service monetary costs. They affirm the near-optimal efficiency of the multi-agent rollout mechanism with decreased execution time.

Acknowledgements The work presented is supported by the EU Horizon 2020 research and innovation program under grant agreement No. 101017171 in the context of the MARSAL project and by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "2nd

Call for H.F.R.I. Research Projects to support Faculty Members & Researchers" (Project Number: 04596).

Funding Open access funding provided by HEAL-Link Greece. The work presented in this paper was supported by the EU Horizon 2020 research and innovation program under grant agreement No. 101017171 within the context of the MARSAL project. Additionally, support was provided by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "2nd Call for H.F.R.I. Research Projects to support Faculty Members & Researchers" (Project Number: 04596).

Data availability The data supporting the findings of this study are available upon request. However, the algorithm developed and used in this research is proprietary/confidential and cannot be shared at this time.

Declarations

Conflict of interest The authors declare that this work was supported by the EU Horizon 2020 research and innovation program in the context of the MARSAL project, under grant agreement No. 101017171. Polyzois Soumplis, Georgios Kontos, and Emmanuel Varvarigos received support by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "2nd Call for H.F.R.I. Research Projects to support Faculty Members & Researchers" (Project Number: 04596). The funding bodies had no role in the design of the study, the collection, analysis, and interpretation of data, or in writing the manuscript. There are no other relationships or activities that could appear to have influenced the submitted work.

Consent for publication All authors have agreed to the publication of this work.

References

1. Villamizar M, Garcés O, Castro H, Verano M, Salamanca L, Casallas R, Gil S. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In: 2015 10th Computing Colombian Conference (10CCC); 2015. p. 583–90. https://doi.org/10.1109/ColumbianCC.2015.7333476
2. Akbar MS, Hussain Z, Sheng QZ, Mukhopadhyay S. 6g survey on challenges, requirements, applications, key enabling technologies, use cases, ai integration issues and security aspects; 2022.
3. Dangi R, Lalwani P, Choudhary G, You I, Pau G. Study and investigation on 5g technology: A systematic review. Sensors (Basel, Switzerland). 2021;22. https://doi.org/10.3390/s22010026.
4. Wang S, Guo Y, Zhang N, Yang P, Zhou A, Shen X. Delay-aware microservice coordination in mobile edge computing: a reinforcement learning approach. IEEE Trans Mob Comput. 2021;20(3):939–51. https://doi.org/10.1109/TMC.2019.2957804.
5. Bernstein D. Containers and cloud: from lxc to docker to kubernetes. IEEE Cloud Comput. 2014;1:81–4. https://doi.org/10.1109/MCC.2014.51.
6. Ren Z, Wang W, Wu G, Gao C, Chen W, Wei J, Huang T. Migrating web applications from monolithic structure to microservices architecture. Assoc Comput Mach. 2018. https://doi.org/10.1145/3275219.3275230.
7. Shi W, Cao J, Zhang Q, Li Y, Xu L. Edge computing: vision and challenges. IEEE Internet Things J. 2016;3:637–46. https://doi.org/10.1109/JIOT.2016.2579198.
8. Goethals T. Fledge: Kubernetes compatible container orchestration on low-resource edge devices
9. Li X, Lian Z, Qin X, Jie W. Topology-aware resource allocation for iot services in clouds. IEEE Access. 2018;6:77880–9. https://doi.org/10.1109/ACCESS.2018.2884251.
10. Kiran N, Liu X, Wang S, Changchuan Y. Vnf placement and resource allocation in sdn/nfv-enabled mec networks; 2020.
11. Silva RACD, Fonseca NLSD. Resource allocation mechanism for a fog-cloud infrastructure, vol. 2018-May. New Jersey: Institute of Electrical and Electronics Engineers Inc.; 2018. https://doi.org/10.1109/ICC.2018.8422237.
12. Sartzetakis I, Soumplis P, Pantazopoulos P, Katsaros KV, Sourlas V, Varvarigos EM. Resource allocation for distributed machine learning at the edge-cloud continuum. ICC 2022 - IEEE International Conference on Communications; 2022. https://doi.org/10.1109/icc45855.2022.9838647
13. Kumar D, Maurya AK, Baranwal G. Chapter 6 - iot services in healthcare industry with fog/edge and cloud computing. In: Singh, S.K., Singh, R.S., Pandey, A.K., Udmale, S.S., Chaudhary, A. (eds.) IoT-Based Data Analytics for the Healthcare Industry. Intelligent Data-Centric Systems, Academic Press; 2021. p. 81–103. https://doi.org/10.1016/B978-0-12-821472-5.00017-X.
14. Sangaiah AK, Pham H, Qiu T, Muhammad K. Convergence of deep machine learning and parallel computing environment for bio-engineering applications. Concurrency and Computation: Practice and Experience. 2019;32(1). https://doi.org/10.1002/cpe.5424.
15. Mahmud R, Srirama SN, Ramamohanarao K, Buyya R. Quality of experience (qoe)-aware placement of applications in fog computing environments. Journal of Parallel and Distributed Computing. 2019;132:190–203. https://doi.org/10.1016/j.jpdc.2018.03.004.
16. Alfakih T, Hassan MM, Gumaei A, Savaglio C, Fortino G. Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa. IEEE Access. 2020;8:54074–84. https://doi.org/10.1109/ACCESS.2020.2981434.
17. Chen L, Xu Y, Lu Z, Wu J, Gai K, Hung PCK, Qiu M. Iot microservice deployment in edge-cloud hybrid environment using reinforcement learning. IEEE Internet Things J. 2021;8:12610–22. https://doi.org/10.1109/JIOT.2020.3014970.
18. Yang C, Xu H, Fan S, Cheng X, Liu M, Wang X. Efficient resource allocation policy for cloud edge end framework by reinforcement learning. In: 2022 IEEE 8th International Conference on Computer and Communications (ICCC); 2022. https://doi.org/10.1109/iccc56324.2022.10065844
19. Wu H, Hua X, Li Z, Ren S. Resource and instance hour minimization for deadline constrained dag applications using computer clouds. IEEE Trans Parallel Distrib Syst. 2016;27(3):885–99. https://doi.org/10.1109/TPDS.2015.2411257.
20. Kliazovich D, Pecero JE, Tchernykh A, Bouvry P, Khan SU, Zomaya AY. Ca-dag: Communication-aware directed acyclic graphs for modeling cloud computing applications. In: Proceedings of the 2013 IEEE Sixth International Conference on Cloud Computing.

CLOUD '13, IEEE Computer Society, USA; 2013. p. 277–284. https://doi.org/10.1109/CLOUD.2013.40.
21. Convolbo MW, Chou J. Cost-aware dag scheduling algorithms for minimizing execution cost on cloud resources. J Supercomput. 2016;72(3):985–1012. https://doi.org/10.1007/s11227-016-1637-7.
22. Kontos G, Soumplis P, Kokkinos P, Varvarigos E. Cloud-Native Applications' Workload Placement over the Edge-Cloud Continuum. In: Proceedings of the 13th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER, SciTePress; 2023;p. 57–66. https://doi.org/10.5220/0011850100003488. INSTICC
23. Sallam G, Ji B. Joint placement and allocation of vnf nodes with budget and capacity constraints; 2019.
24. Bertsekas D. Multiagent reinforcement learning: rollout and policy iteration. IEEE/CAA J Autom Sin. 2021;8(2):249–72. https://doi.org/10.1109/JAS.2021.1003814.
25. Hadary O, Marshall L, Menache I, Pan A, Greeff EE, Dion D, Dorminey S, Joshi S, Chen Y, Russinovich M, Moscibroda T. Protean: VM allocation service at scale. In: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), USENIX Association; 2020. p. 845–861. https://www.usenix.org/conference/osdi20/presentation/hadary
26. Mobile Experts Inc.: EDGE INSIGHT: Cost of Outpost vs DIY Edge Cloud; 2020.
27. Rutlege K. Bandwidth Economics are the business case for Edge Computing. LinkedIn; 2019. https://shorturl.at/hnqs4
28. Madden J. Analysis : The economics of edge computing; 2020.
29. Alibaba Group: Alibaba Cluster Data 2017. Accessed: [12/08/2023] (2017). https://github.com/alibaba/clusterdata/tree/master/cluster-trace-v2017