POSTPRINT VERSION. The final version is published here : Gouaouri, M. D. E., Ouahouah, S., Bagaa, M., Ouameur, M. A., Daniel, M. et Ksentini, A. (2026). Diktopos: A two-stage framework for joint container-based microservice placement and distributed volume allocation on cloud-edge networks. IEEE Transactions on Cloud Computing. https://doi.org/10.1109/TCC.2026.3676203. CC BY 4.0.

A two-stage framework for topology-aware joint microservice placement and distributed volume allocation on cloud-edge networks

Mohammed Dhiya Eddine Gouaouri*, Sihem Ouahouah†, Miloud Bagaa*,
Messaoud Ahmed Ouameur*, Adlen Ksentini‡
*Dept. of Electrical and Computer Engineering, Université du Québec à Trois-Rivières, Canada
{mohammed.dhyia.eddine.gouaouri, miloud.bagaa, messaoud.ahmed.ouameur}@uqtr.ca
†Dept. of Communications and Networking, Aalto University, Espoo, Finland
sihem.ouahouah@aalto.fi
‡EURECOM, Campus SophiaTech, France
adlen.ksentini@eurecom.fr

***Abstract*—The Cloud Edge Continuum enables the deployment of latency-sensitive and data-intensive applications closer to end users, but it poses challenges for microservice placement due to resource heterogeneity and limited edge capacity, especially when storage requirements must be met through aggregated node resources. To address this, we propose a two-stage, topology-aware optimization framework that jointly handles microservice deployment and distributed storage volume allocation in edge networks. Our framework decomposes this joint placement problem into two subproblems, microservice placement followed by a distributed volume allocation subproblem, with the goal of optimizing computation, communication, energy, and storage costs. At its core is a lightweight, rank-based heuristic that ensures scalable, accurate placement across distributed edge nodes. Evaluations on real-world scenarios show our method achieves near-optimal placement (within 1.67% of the exact solution), reduces system costs by up to 30%, and accelerates convergence by 5× compared to state-of-the-art approaches, demonstrating its suitability for dynamic, resource-constrained edge environments.**

***Index Terms*—Cloud-Edge Computing Continuum, Microservice Placement, Optimization, Distributed Storage Virtualization**

I. Introduction

In recent years, the emergence of the Cloud-Edge Continuum has transformed the landscape of distributed computing. Traditional cloud-centric models, while powerful, often struggle to meet the stringent latency and bandwidth requirements of modern applications. This has led to a paradigm shift in which computation and storage are increasingly being pushed toward the edge of the network, closer to data sources [1]. This shift enables new classes of latency-sensitive and data-intensive applications, such as autonomous driving, augmented and virtual reality (AR/VR), and real-time video analytics, to be executed more efficiently. By reducing end-to-end delay and alleviating core network congestion, edge computing provides faster responses and better performance guarantees. However, the edge network is fundamentally different from the centralized clouds. Edge environments are geographically distributed, heterogeneous, and resource-constrained compared to the cloud. They must cope with challenges, such as device mobility, network variability, failures, and limited compute/storage capacity. To address these challenges, system designers are increasingly adopting a microservice-based architecture, where monolithic applications are decomposed into loosely coupled, independently deployable components called microservices. This design pattern enhances scalability, fault tolerance, and deployment flexibility, making it well-suited for the dynamic and decentralized nature of edge computing environments. Still, the heterogeneity and fluctuating resource availability of edge networks complicate the deployment of microservices with diverse requirements and communication dependencies [2]. For instance, compute-intensive services like image recognition differ greatly from data and network-intensive ones like video streaming. Poor placement decisions, especially those neglecting storage needs can increase latency, energy usage, and costs, while degrading service quality. This highlights the need for deployment strategies that are latency-aware, energy-efficient, and resource-aware.

While much work has aimed to optimize edge deployment [2]–[6], most of them focus on latency or compute resource efficiency, often neglecting energy consumption, a crucial objective and constraint in edge-computing systems. Moreover, to the best of our knowledge, none of the state-of-the-art works tackled the joint problem of microservice placement and distributed storage allocation as it is considered an indivisible resource. Edge servers often lack sufficient local storage, making it necessary to distribute data across nodes. Intelligent strategies are needed not just for service placement, but for virtualizing and distributing storage in a consistent way. To address these challenges, we propose a two-stage framework that jointly optimizes microservice placement and distributed storage allocation. Our approach enables edge nodes to collaboratively form a virtualized storage layer, providing each microservice with the storage capacity it needs. The goal is to minimize total deployment

cost, combining compute, communication, latency, and energy factors.

The rest of this paper is organized as follows: Section II reviews related research. Section III details the problem formulation and methodology. Section IV presents an extensive evaluation, and Section V concludes the paper.

II. Related Works

Microservice deployment in cloud-edge environments can be broadly classified into two main categories: model-based and learning-based approaches.

Model-based methods formulate placement as an optimization problem, solved via heuristics or metaheuristics. For instance, The solutions [7] and [8] use Particle Swarm and Whale Optimization to minimize energy and latency, but overlook complex application topologies. To handle structured applications, authors in [9] introduce an Ant Colony-based scheduler, though limited to tree topologies. In contrast, the solution proposed in [10], optimizes placement using graph-based strategies and Kubernetes-level enhancements, considering latency and bandwidth. The approach in [11] further maps service-cluster dependencies heuristically. Despite progress, most of these approaches ignore storage constraints and data locality, limiting performance in realistic edge environments.

Learning-based methods, particularly those leveraging deep reinforcement learning (DRL), aim to learn adaptive placement policies. [12] uses Deep Q-Learning for 5G, while [13] addresses multi-objective placement for load balancing and reduced communication. More recent work, such as [14] and [2], integrate GNNs to better model microservice dependencies. However, these approaches often lack generalizability to dynamic edge environments and typically ignore distributed storage needs, an essential factor for data-intensive services.

In contrast, our work is the first attempt to jointly optimize microservice placement and distributed volume allocation in a model-based framework. This approach ensures efficient resource usage, data locality, and low-latency service delivery across heterogeneous cloud-edge infrastructures.

III. Problem Formulation and Methodology

In this section, we define the problem of topology-aware joint microservice and distributed volume allocation as an Integer Nonlinear Programming (INLP) problem. Next, we introduce a low-complexity, two-stage rank-based algorithm to solve the problem efficiently.

A. Problem formulation

We model the microservices-based application as a directed graph in $\mathcal{G}_{\mathcal{M}} = (\mathcal{V}_{\mathcal{M}}, \mathcal{E}_{\mathcal{M}}, \mathcal{X}_{\mathcal{M}}, \mathcal{F}_{\mathcal{M}})$ in which $\mathcal{V}_{\mathcal{M}}$ represents the set of microservices of the application, $\mathcal{E}_{\mathcal{M}}$ is the set of edges that connect microservices $\mathcal{V}_{\mathcal{M}}$ representing the invocation relationships. $\mathcal{X}_{\mathcal{M}}$ is the set of node features (CPU demands denoted as cpu_m, memory demands mem_m, and disk requirements $disk_m$). Meanwhile, $\mathcal{F}_{\mathcal{M}}$ represents the set of edge features.

The edge-computing infrastructure is modeled as a graph $\mathcal{G}_{\mathcal{N}} = (\mathcal{V}_{\mathcal{N}}, \mathcal{E}_{\mathcal{N}}, \mathcal{X}_{\mathcal{N}}, \mathcal{F}_{\mathcal{N}})$, whereby $\mathcal{V}_{\mathcal{N}}$ is the set of edge servers, $\mathcal{E}_{\mathcal{N}}$ is the set of network connections between edge servers, $\mathcal{X}_{\mathcal{N}}$ is the set of edge server node features (available CPU cpu_n, CPU frequency f_n, per CPU unit power consumption $pcpu_n$, available memory mem_n and attached storage size $disk_n$), $\mathcal{F}_{\mathcal{N}}$ is the set of network link features which contain the link bandwidth $\mathcal{F}_{\mathcal{N}} = \{bw(e_i, e_j) \mid (e_i, e_j) \in \mathcal{E}_{\mathcal{N}}\}$.

Let $\mathcal{P}(e_i, e_j)$ denote the network routing path function between edge server e_i and e_j. In this work, we consider a shortest-path based request routing. $\mathcal{P}(e_i, e_j) = \{(e_i, e_1), ..., (e_k, e_j)\}$.

We define the binary variable $x_{m,n} \in \{0, 1\}$ such that:

$$x_{m,n} = \begin{cases} 1, & \text{if microservice } m \text{ is deployed on edge server } n \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

We also define the variable $y_{m,n} \in \mathbb{N}$ to denote the size of the disk allocated to the service m on the edge server n.

a) Placement computation cost: The computation cost is defined as the worst-case application execution time, which is defined as follows:

$$L_{compute} = \sum_{m \in \mathcal{V}_{\mathcal{M}}} \sum_{e \in \mathcal{V}_{\mathcal{N}}} x_{m,e} \cdot \frac{cpu_m}{f_e} \tag{2}$$

By minimizing this cost, we encourage the model to place compute-heavy microservices on high-frequency servers.

b) Microservice to Microservice communication cost: The communication cost measures the overhead associated with inter-microservice communication.

$$L_{comm} = \sum_{m_i, m_j \in \mathcal{V}_{\mathcal{M}}} \sum_{e_i, e_j \in \mathcal{V}_{\mathcal{N}}} x_{m_i, e_i} \cdot x_{m_j, e_j} \cdot \sum_{(u,v) \in \mathcal{P}(e_i, e_j)} \frac{bw_{min}(m_i, m_j)}{bw(u, v)} \tag{3}$$

Here, $bw_{min}(m_i, m_j)$ denotes the minimum bandwidth required for efficient communication between microservices m_i and m_j.

The formula penalizes deployment decisions where communicating microservices are placed on servers connected via low-bandwidth or longer paths. This encourages co-locating tightly coupled microservices or routing traffic through high-bandwidth links, aligning with the goal of minimizing latency and network bottlenecks.

c) Energy consumption cost: As previously mentioned, each edge server $e_n \in \mathcal{V}_{\mathcal{N}}$ is associated with a CPU power consumption rate denoted by $pcpu_{e_n}$ (i.e., power consumed per unit of CPU utilization). The energy consumption cost reflects the total power consumed by all deployed microservices based on their CPU demands and the power profile of the hosting edge servers. The total energy consumption cost is computed as:

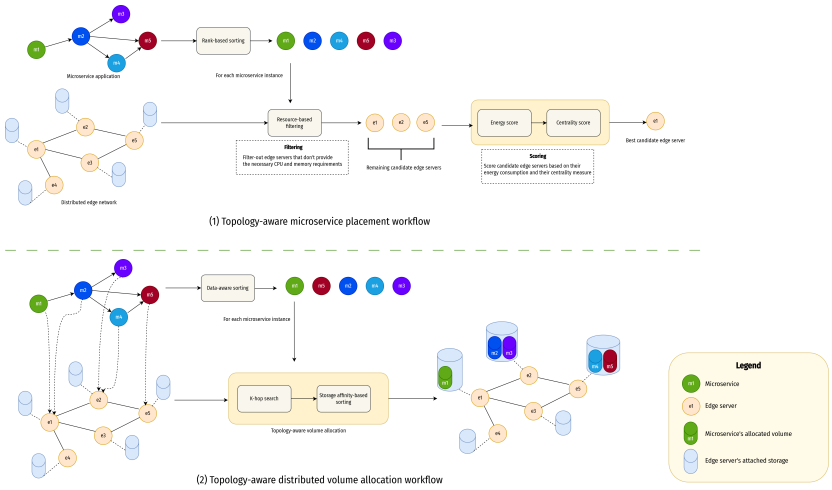

Figure 1: Joint microservice placement and distributed volume allocation

$$L_{energy} = \sum_{m \in \mathcal{V}_\mathcal{M}} \sum_{e \in \mathcal{V}_\mathcal{N}} x_{m,e} \cdot cpu_m \cdot pcpu_e \tag{4}$$

d) Data volume allocation cost: In microservice-based applications, services often require access to persistent or shared data volumes. In cloud-edge environments, these data volumes might not reside locally on the same node as the microservice. Instead, the data may be distributed across multiple edge or cloud nodes and accessed over the network. This incurs communication overhead depending on the data location and network conditions.

The cost associated with allocating and accessing these distributed data volumes is formulated in 5. The formula encourages placing data volumes close to the microservice or on high-bandwidth paths, minimizing data access latency and network congestion. It also discourages distributing the data volume unnecessarily across poorly connected nodes

$$L_{vol} = \sum_{m \in \mathcal{V}_\mathcal{M}} \sum_{e_i \in \mathcal{V}_\mathcal{N}} \sum_{e_j \in \mathcal{V}_\mathcal{N}} x_{m,e_i} \cdot \sum_{(u,v) \in \mathcal{P}(e_i,e_j)} \frac{y_{m,e_j}}{bw(u,v)} \tag{5}$$

Hence, the total cost is defined as the sum of all costs:

$$L_{tot} = \alpha \cdot L_{compute} + \beta \cdot L_{comm} + \gamma \cdot L_{vol} + \delta \cdot L_{energy} \tag{6}$$

Here α, β, γ and δ are the weighting parameters of the objectives.

The goal is to solve the following optimization problem:

$$\text{minimize} \quad L_{tot} \tag{7}$$

$$\text{subject to} \quad \sum_{n \in \mathcal{V}_\mathcal{N}} x_{m,n} = 1, \quad \forall m \in \mathcal{V}_\mathcal{M} \tag{C1}$$

$$\sum_{m \in \mathcal{V}_\mathcal{M}} x_{m,n} \cdot cpu_m \leq cpu_n, \quad \forall n \in \mathcal{V}_\mathcal{N} \tag{C2}$$

$$\sum_{m \in \mathcal{V}_\mathcal{M}} x_{m,n} \cdot mem_m \leq mem_n, \quad \forall n \in \mathcal{V}_\mathcal{N} \tag{C3}$$

$$\sum_{m \in \mathcal{V}_\mathcal{M}} y_{m,n} \leq disk_n, \quad \forall n \in \mathcal{V}_\mathcal{N} \tag{C4}$$

$$\sum_{n \in \mathcal{V}_\mathcal{N}} y_{m,n} = d_m, \quad \forall m \in \mathcal{V}_\mathcal{M} \tag{C5}$$

$$y_{m,n} \geq 0 \text{ and } x_{m,n} \in \{0,1\}, \quad \forall m \in \mathcal{V}_\mathcal{M}, n \in \mathcal{V}_\mathcal{N} \tag{C6}$$

The constraint C1 ensures that each microservice is deployed on exactly one edge server node. Meanwhile, constraints C2 and C3 make sure that requested resources for microservices deployed on any edge server don't exceed available CPU and memory resources of that server. Constraint C4 ensures that allocated storage on any edge server does not exceed the total storage attached to that server. Constraint C5 ensures that the requested storage is distributed on the attached storage available on the edge servers. Last but not least, constraint C6 ensures that the variables are binary integers.

The formulated problem represents an Integer Nonlinear Program (INLP), which is computationally challenging to solve optimally, especially for large-scale microservice applications and edge networks. To address this complexity, we propose a novel two-stage topology-aware rank-based heuristic that jointly optimizes microservice deployment and storage allocation. Our approach achieves near-optimal solutions with significantly lower computational overhead. In the next subsection, we present our solution.

B. Near-Optimal Rank-Based Joint Microservice Placement and Volume Allocation on Distributed Edge Network

In this section, we present our proposed algorithms for solving the previous problem efficiently. The joint placement problem is decomposed into two sub-problems, the microservice placement and distributed volume allocation, as presented in Figure 1.

a) Microservice placement: To find the microservice placement decision, we follow the *Sort*, *Filter*, and *Score* pattern employed by multiple research works, such as in [10] and developed in real microservice orchestration systems, such as Kubernetes.

Definition 1 (microservice criticality rank): The criticality rank $R(m)$ for microservice $m \in \mathcal{V}_\mathcal{M}$ defines how important a microservice m is and it is computed recursively as:

$$R(m) = \overline{w}_m + \psi(m) + \max_{n \in \mathcal{N}(m)} (\overline{c}_{m,n} + R(n)) \tag{8}$$

Where:

- $\overline{w}_m = \frac{1}{\mathcal{V}_\mathcal{N}} \sum_{e \in \mathcal{V}_\mathcal{N}} \frac{cpu_m}{f_e}$ represents the average computation cost of microservice m when deployed on the edge-computing network.
- $\psi(m) = \frac{deg(m)}{max_{k \in \mathcal{V}_\mathcal{M}}(deg(k))}$ represents the graph degree centrality of microservice node m.
- $\overline{c}_{m,n} = \mathbb{E}[\sum_{(u,v) \in \mathcal{P}(e^*(m), e^*(n))} \frac{bw_{min}(m,n)}{bw(u,v)}]$ represents the expected communication cost between microservice m and n. It estimates the average network cost incurred when microservices m and n, which interact during runtime, are deployed on different edge nodes $e^*(m)$ and $e^*(n)$.

Algorithm 1 Microservice Placement

Input: $\mathcal{G}_\mathcal{M} = (\mathcal{V}_\mathcal{M}, \mathcal{E}_\mathcal{M})$ (microservice graph) $\mathcal{G}_\mathcal{N} = (\mathcal{V}_\mathcal{N}, \mathcal{E}_\mathcal{N})$ (edge network) Resource demands $\{cpu_m, mem_m\}_{m \in \mathcal{V}_\mathcal{M}}$ Server capacities $\{cpu_e, mem_e, f_e\}_{e \in \mathcal{V}_\mathcal{N}}$

Output: Placement matrix $X \in \{0,1\}^{|\mathcal{V}_\mathcal{M}| \times |\mathcal{V}_\mathcal{N}|}$

foreach *microservice* $m \in \mathcal{V}_\mathcal{M}$ **do**
  Compute $R(m) \leftarrow \overline{w}_m + \psi(m) + \max_{n \in \mathcal{N}(m)} (\overline{c}_{m,n} + R(n))$
Sort $\mathcal{V}_\mathcal{M}$ by $R(m)$ in descending order

foreach *microservice* $m \in$ *sorted* $\mathcal{V}_\mathcal{M}$ **do**
  $\mathcal{F} \leftarrow \{e \in \mathcal{V}_\mathcal{N} \mid cpu_e \geq cpu_m \wedge mem_e \geq mem_m\}$
  if $\mathcal{F} = \emptyset$ **then**
    Place m on the cloud **continue**
  // $\xi(e)$ is the edge server's betweenness centrality
  $e^* \leftarrow_{e \in \mathcal{F}} \left[\alpha \cdot \xi(e) + \beta \cdot \left(1 - \frac{energy(e)}{max\_energy}\right)\right]$
  $X[m, e^*] \leftarrow 1$
  $cpu_{e^*} \leftarrow cpu_{e^*} - cpu_m$
  $mem_{e^*} \leftarrow mem_{e^*} - mem_m$
return X

The main steps of the proposed solution are summarized in Algorithm 1. The microservice placement algorithm (Algorithm 1) follows a two-stage approach. First, each microservice is assigned a criticality rank based on its computational load, graph centrality, and dependency-aware communication cost. Microservices are then placed in descending order of rank. For each, eligible edge servers are filtered based on resource availability. The best-fit server is selected using a scoring function that considers betweenness centrality and energy efficiency. Resources are updated post-placement. If no server satisfies the resource demands, the algorithm flags infeasibility.

b) Distributed volume allocation: Once the microservices have been placed, the algorithm proceeds to allocating the data volumes on the distributed storage.

Definition 2: (Storage affinity) Storage affinity is a metrics that defines the *preference* for placing a microservice's m requested data blocks (requested volume) on a particular edge server, based on three factors:

1) Network proximity to the microservice's host server
2) Available bandwidth along the routing path.
3) Storage capacity of the target server.

It's defined as follows:

$$A(m,e) = \begin{cases} 1 & \text{if } e = e^*(m) \\ \frac{disk_e}{disk_{max}} \cdot \sum_{(u,v) \in \mathcal{P}(e^*(m), e)} \frac{bw(u,v)}{bw_{max}} & \text{otherwise} \end{cases} \tag{9}$$

Where:

- $e^*(m)$ is the edge server on which the microservice m has been deployed.
- $disk_e$ is the storage available attached to the server e.
- bw_{max} and $disk_{max}$ are normalization factors.

The affinity metric prioritizes high-bandwidth paths and reduces network latency by colocating data with the microservices that requested it.

The distributed volume allocation algorithm (Algorithm 2) assigns data blocks to edge or cloud servers based on storage affinity. This metric considers proximity to the microservice's host, bandwidth along the routing path, and available storage. The algorithm prioritizes microservices with high data and bandwidth needs, first attempting local storage to maximize data locality. If local resources are insufficient, it explores nearby servers within a hop limit, selecting those with the highest affinity. Cloud storage is used only as a fallback. This strategy ensures efficient, latency-aware data placement with minimal overhead.

Algorithm 2 Distributed Volume Allocation

Input: Placement matrix X, Microservice demands $\{d_m, bw_m\}$, Server storage $\{disk_e\}$, Network bandwidth $\{bw(u,v)\}$, Maximum hops k_{max}

Output: Allocation matrix $Y \in \mathbb{N}^{|\mathcal{V}_\mathcal{M}| \times |\mathcal{V}_\mathcal{N}|}$

Sort $\mathcal{V}_\mathcal{M}$ by $d_m \cdot bw_m$ (descending)
foreach *microservice* $m \in$ *sorted* $\mathcal{V}_\mathcal{M}$ **do**
  $r \leftarrow d_m$ $e^*(m) \leftarrow$ server where $X[m, e^*(m)] = 1$
  for $k \leftarrow 0$ **to** k_{max} **do**
    $\mathcal{N}_k \leftarrow k$-hop neighbors of $e^*(m)$ Sort $\mathcal{N}_k$ by storage affinity $A(m,e)$
    foreach *server* $e \in \mathcal{N}_k$ **do**
      if $r > 0$ *and* $disk_e > 0$ **then**
        $alloc \leftarrow \min(r, disk_e)$; $Y[m,e] \leftarrow alloc$
        $disk_e \leftarrow disk_e - alloc$; $r \leftarrow r - alloc$
    if $r = 0$ **then**
      Break
  if $r > 0$ **then**
    Allocate r on cloud servers
return Y

IV. Experimentation

This section presents the simulation environment of the experiments conducted to evaluate the performance of our proposed system. We first present the simulated deployment infrastructure, Next, the testbed application used to evaluate and validate the system. Then, we present the baseline approaches

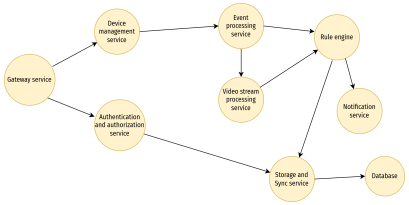

Figure 2: Smart Home Application Microservice Graph

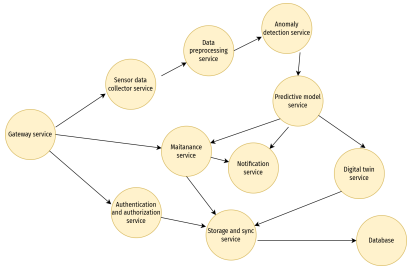

Figure 3: Industrial IoT Predictive Maintenance Application Graph

to compare our system with. Lastly, performance analysis along with scalability analysis to assess the performance capability of our proposed approach for large microservice applications and large infrastructure networks.

A. Deployment Infrastructure

We simulate an edge computing infrastructure comprising 10 nodes interconnected by 25 substrate links. Following [2], [15], we model the topology as a Waxman random graph, a well-established model for edge networks [16], [17].

Each edge server provides three resource types:

- Computational resources (CPU)
- Memory capacity
- Attached disk storage (for microservice volume allocation)

Notably, storage volumes can be distributed across multiple edge servers while appearing as a unified resource to microservices. Resource capacities are sampled uniformly from the ranges specified in Table I.

Table I: Resource Constraints

Resource	Min	Max
CPU	2000 (CPU unit)	8000 (CPU unit)
Memory	4 (GB)	8(GB)
Storage	4 (GB)	8(GB)

Similarly, the bandwidth of the links between edge servers is generated uniformly at random between 500 Mb/s and 1000 Mb/s.

B. Testbed applications

Two microservice applications are simulated and deployed to evaluate the performance of the scheduling software: i) *Smart home application*; ii) *Industrial IoT predictive maintenance application*. The applications are designed to be distributed, latency-sensitive and involve multiple interdependent microservices that interact with each other and rely on timely data processing close to end devices, making them an attractive real-world scenario for testing the placement algorithms.

The resource requirements (CPU, memory, and storage) of each microservice in each application are randomly sampled uniformly. Similarly, the minimum bandwidth requirement of each invocation link is uniformly sampled between 10 Mb/s and 50 Mb/s.

a) Smart Home Application: The Smart Home IoT application leverages a modular microservice architecture deployed on edge servers to manage and automate various home devices and sensors in real time as shown in Figure 2. The application models a typical system to manage smart homes in an intelligent area. In this application, microservice resources (CPU, memory and requested storage) are uniformly sampled at random depending on whether the microservice is compute-heavy, memory-heavy or disk-heavy such as the database

b) IIOT predictive Maintenance Application: The Industrial IoT Predictive Maintenance application is designed to monitor, analyze, and optimize the operational health of industrial equipment using a distributed microservice architecture deployed across edge computing nodes. It components graph is shown in Figure 3. This application models an intelligent system to manage and maintain industrial IoT devices. Similar to the previous application, microservice resources (CPU, memory, and requested storage) are selected using uniform distribution.

C. Comparison Baseline Approaches

To evaluate the robustness of our proposed algorithm, we compare it against established methods from the literature.

a) Exact Approach: The original INLP model was transformed into an equivalent Integer Linear Programming (ILP) formulation through a series of mathematical transformations. Owing to space limitations, we do not provide the full details of the ILP formulation. The resulting ILP was solved to optimality using the Gurobi optimizer. All experiments were performed on a system with an 8-core Intel i5-13420H CPU running at 2.10 GHz and equipped with 16 GB of RAM.

b) GA-Based Approach: Genetic Algorithm (GA) is a metaheuristic inspired by natural selection. It iteratively refines a population of solutions via crossover and mutation to evolve high-quality placements.

c) PSO-Based Approach: Particle Swarm Optimization (PSO) is inspired by swarm intelligence, where particles (solutions) explore the search space. We adopt a discrete PSO variant tailored for microservice-to-server mapping, using probabilistic modeling for valid assignments.

d) Topology-Based Approach: Based on [10], this method uses topological sorting (e.g., Tarjan's algorithm) to order microservices, followed by a filter-and-score process that considers resource availability and bandwidth for server selection.

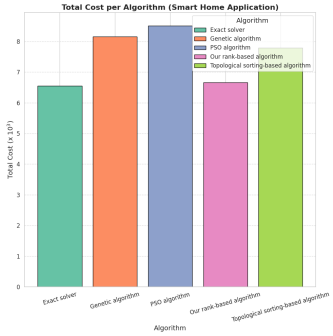

Figure 4: Total Cost Comparison For Smart Home Application

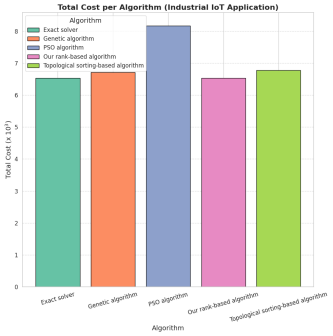

Figure 5: Total Cost Comparison For Industrial IoT Application

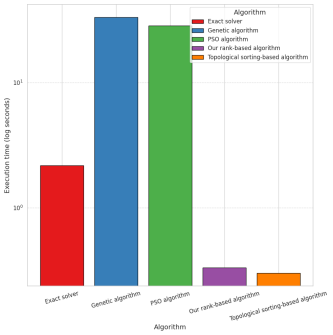

Figure 6: Convergence Time Comparison For Smart Home Application

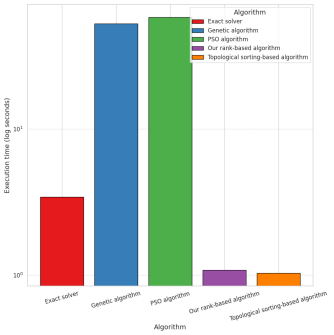

Figure 7: Convergence Time Comparison For Industrial IoT Application

D. Results and Analysis

a) Cost Analysis: As shown in Figures 4 and 5, our rank-based heuristic achieves near-optimal placement costs, within 1.66% for Smart Home and only 0.05% for Industrial IoT scenarios, relative to the optimal solution obtained using Gurobi solver. In contrast, GA and PSO yield up to 30% higher costs, while the topological-sorting-based method results in deviations over 3%. These results underline the cost-efficiency of our approach, which consistently outperforms heuristic baselines and closely tracks the optimal.

b) Convergence Time Analysis: Figures 6 and 7 demonstrate that our method converges faster than GA and PSO, which require prolonged iterative optimization. On average, our method achieves convergence in 4 seconds, compared to 20 seconds for GA and 22 seconds for PSO. Although the state-of-the-art topological-sorting-based heuristic converges in under 1 second, it does so at the cost of reduced placement quality. Compared to the exact solver, our method is approximately 5× faster, while maintaining near-optimal accuracy, making it well suited for deployment in dynamic edge environments.

V. Conclusion

This paper tackles joint microservice placement and distributed storage allocation at the edge with a two-stage, topology-aware framework. A rigorous cost model paired with a lightweight rank-based heuristic delivers near-optimal placement (within 1.67% to optimal solution) while cutting execution time versus baselines. The approach scales to dynamic edge settings, and future work will explore adaptive strategies, reinforcement learning, and multi-objective optimization.

References

[1] D. Rosendo, A. Costan, P. Valduriez, and G. Antoniu, "Distributed intelligence on the edge-to-cloud continuum: A systematic literature review," *Journal of Parallel and Distributed Computing*, vol. 166, pp. 71–94, 2022.

[2] S. Chen, Q. Yuan, J. Li, H. He, S. Li, X. Jiang, and J. Yang, "Graph neural network aided deep reinforcement learning for microservice deployment in cooperative edge computing," *IEEE Transactions on Services Computing*, 2024.

[3] P. Kayal and J. Liebeherr, "Autonomic service placement in fog computing," in *2019 IEEE 20th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 2019, pp. 1–9.

[4] S. Pallewatta, V. Kostakos, and R. Buyya, "Microservices-based iot application placement within heterogeneous and resource constrained fog computing environments," in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, 2019, pp. 71–81.

[5] 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*, vol. 147, p. 102441, 2019.

[6] M. Hu, H. Wang, X. Xu, J. He, Y. Hu, T. Deng, and K. Peng, "Joint optimization of microservice deployment and routing in edge via multi-objective deep reinforcement learning," *IEEE Transactions on Network and Service Management*, 2024.

[7] 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 *2019 18th international symposium on parallel and distributed computing (ISPDC)*. IEEE, 2019, pp. 32–40.

[8] M. Ghobaei-Arani and A. Shahidinejad, "A cost-efficient iot service placement approach using whale optimization algorithm in fog computing environment," *Expert Systems with Applications*, vol. 200, p. 117012, 2022.

[9] P. Han, Y. Liu, and L. Guo, "Interference-aware online multicomponent service placement in edge cloud networks and its ai application," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 557–10 572, 2021.

[10] J. Santos, C. Wang, T. Wauters, and F. De Turck, "Diktyo: Network-aware scheduling in container-based clouds," *IEEE Transactions on Network and Service Management*, vol. 20, no. 4, pp. 4461–4477, 2023.

[11] X. Li, J. Zhou, X. Wei, D. Li, Z. Qian, J. Wu, X. Qin, and S. Lu, "Topology-aware scheduling framework for microservice applications in cloud," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 5, pp. 1635–1649, 2023.

[12] S. Yu, X. Chen, Z. Zhou, X. Gong, and D. Wu, "When deep reinforcement learning meets federated learning: Intelligent multitimescale resource management for multiaccess edge computing in 5g ultradense network," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2238–2251, 2020.

[13] W. Lv, Q. Wang, P. Yang, Y. Ding, B. Yi, Z. Wang, and C. Lin, "Microservice deployment in edge computing based on deep q learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 11, pp. 2968–2978, 2022.

[14] W. Lv, P. Yang, T. Zheng, C. Lin, Z. Wang, M. Deng, and Q. Wang, "Graph-reinforcement-learning-based dependency-aware microservice deployment in edge computing," *IEEE Internet of Things Journal*, vol. 11, no. 1, pp. 1604–1615, 2023.

[15] B. M. Waxman, "Routing of multipoint connections," *IEEE journal on selected areas in communications*, vol. 6, no. 9, pp. 1617–1622, 1988.

[16] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1040–1057, 2020.

[17] M. Naldi, "Connectivity of waxman topology models," *Computer communications*, vol. 29, no. 1, pp. 24–31, 2005.