TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 10/16 pp2485–2503
DOI: 10.26599/TST.2024.9010255
Volume 31, Number 5, October 2026

Research Article

Adaptive Microservice Deployment for Data Partition-Based Applications in MEC

Zhaowu Huang, Fang Dong*, Wenlong Ruan, Xiaolin Guo, and Jinghui Zhang

Abstract: Recently, Mobile Edge Computing (MEC) has used lightweight container-based microservices to provide resources for artificial intelligence applications, which will be decomposed into multiple dependent components, forming a Directed Acyclic Graph (DAG). In MEC, users will partition the input of the computation-intensive tasks into multiple sub-tasks for parallel execution acceleration. To satisfy concurrency, app vendors must deploy multiple container replicas for a microservice. Due to the limited capacity of edge servers, containers need to be deployed into geographically distributed and heterogeneous edge servers, resulting in significant inter-edge server traffic. To this end, we propose an adaptive scheme to guide microservice deployment for data partition-based applications in the MEC. We model the multi-replica microservice deployment problem as an integer programming problem to minimize operation costs. We propose a Deterministic Local Search-based Microservice Deployment algorithm (DLSMD), that chooses a superior neighborhood solution iteratively to solve it. We also formulate a more general problem considering both computing and communication time to minimize the total completion time and devise a Heuristic Microservice Deployment (HMD) algorithm to solve it. Extensive simulation results show that DLSMD and HMD outperform other benchmarks, achieving up to 4× and 3.12× speedups in terms of the inter-server traffic and total completion time reduction, respectively.

Key words: Mobile Edge Computing (MEC); microservice deployment; Directed Acyclic Graph (DAG)

1 Introduction

With the development of end devices and Deep Neural Network (DNN), recent years have witnessed a booming of device artificial intelligence applications, such as AR[1], VR[2], and autonomous driving[3]. To obtain high inference accuracy, these DNN-driven applications bring a large amount of computation, which cannot be executed by the resource-limited device and remote cloud to guarantee low latency. Mobile Edge Computing (MEC)[4–6] is widely adopted as a promising paradigm to pave the last mile for such intelligence applications by providing services in proximity to end devices[7–9]. In MEC, to meet stringent response time requirements, researchers tend to partition input data of computation-intensive tasks to form multiple sub-tasks and distribute them to edge servers for parallel execution acceleration[10–13]. For example, Elf[10] partitions the high-resolution video frames into multiple blocks and offloads these blocks into edge servers for parallel execution, achieving low-latency object detection applications.

Recently, lightweight container-based microservices have been widely advocated as an edge service provision paradigm, due to their lightweight

• Zhaowu Huang, Fang Dong, Wenlong Ruan, Xiaolin Guo, and Jinghui Zhang are with School of Computer Science and Engineering, Southeast University, Nanjing 211189, China. E-mail: zwh@seu.edu.cn; fdong@seu.edu.cn; ruanwenl@seu.edu.cn; xlinguo@seu.edu.cn; jhzhang@seu.edu.cn.
* To whom correspondence should be addressed.
Manuscript received: 2024-05-28; revised: 2024-11-01; accepted: 2024-12-19

advantages, which offer flexibility and elasticity over conventional Virtual Machines (VMs)[14]. In this manner, a monolithic DNN-driven application is decomposed into multiple coupled microservices, which form a Directed Acyclic Graph (DAG)[15, 16]. For example, in Serverless Edge Computing (SEC), an application is decomposed into multiple microservices that are encapsulated as functions and explicitly chained together using tools to compose the entire application, thus forming DAG[17]. As shown in Fig. 1, the autonomous driving application with multiple microservices (e.g., video capture, object tracking, lane detection, and advanced driver assistance) is designed as a DAG workflow. These microservices are then deployed onto edge servers, with each being developed and deployed independently[18–20]. Due to the need for parallel processing of multiple subtasks generated by applications based on data partitioning, application providers must deploy multiple container instances based on the number of subtasks within microservices to prevent task queuing and ensure that users receive low-latency services, as shown in Fig. 1. Since the resources of one edge server are not powerful enough as the data center to host all microservices, the microservices must be deployed into geographically distributed edge servers, generating communication

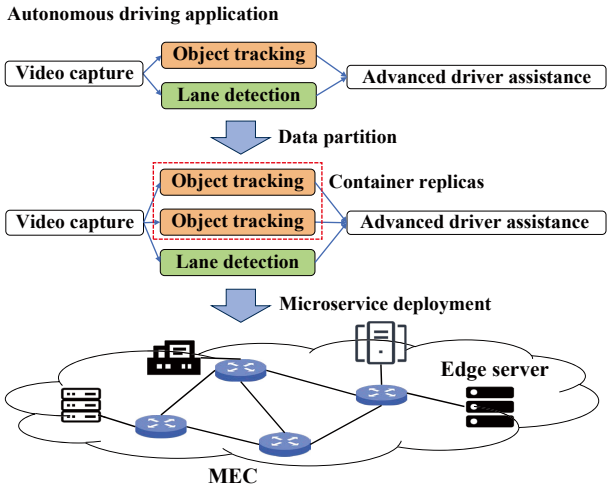


Fig. 1 Illustration of data partition based autonomous driving application. In the microservice architecture, a monolithic application is decomposed into multiple coupled microservices and deployed into edge servers. Users partition the input of the computation-intensive tasks, forming multiple sub-tasks that need to be processed in parallel. Thus, app vendors need to deploy multiple container replicas for the same type of microservices. These container replicas will be deployed into MEC.

overhead caused by data dependency among microservices. Their communication needs to go through the network stacks at the host layer, which needs expensive packet encapsulation and decapsulation operations, and also interferes with other applications, so it induces significant networking latency[20]. Previous investigations[21] have illuminated that the intermediate data transmission of edge-native applications is time-consuming, consuming up to 75% of the overall application completion time. The microservices deployment is a relatively long-term decision compared to computation offloading. To reduce the operation cost of app vendors and improve the performance of applications, it is fundamentally important to minimize the communication traffic among containers at different edge servers.

Implementing microservice deployments with low traffic in MEC poses fundamental challenges. First, heterogeneous edge servers have different capacities, and the microservices of applications constitute different DAG structures[15, 16]. Deploying these microservices with different DAG structures into heterogeneous edge servers leads to resource competition, and deploying each DAG structure independently results in suboptimal performance. Partitioning a DAG into two parts involves complicated graph-theoretic analysis, which may lead to NP-hardness in performance optimization[22]. In this problem, the DAGs may be partitioned into more than two parts due to multiple available edge servers which is more intractable. Second, the data partition based application requires multiple container replicas to be deployed for a microservice in advance to support parallel execution, which will lead to more complex data dependency on the application execution flow. Specifically, for any pair of dependent microservices, all containers of the predecessor and those of the successor need to be connected to each other, which further complicates the microservice deployment problem in MEC. Thus, how to adaptively deploy multi-replica microservices for data partition-based applications in MEC to minimize communication traffic is an intractable and important problem.

Previous microservice deployment works can be categorized into three distinct categories based on the application structure including monolithic applications[23–25], chained-structured applications[26, 27], and DAG-structured applications[28–32]. However, existing work does not consider the effect of data

partition on microservice deployment for multiple DAG-structured applications in MEC, which can not handle the above problem.

In this paper, we propose an adaptive microservice deployment scheme for data partition-based applications in MEC to minimize communication traffic. We first model the multi-replica microservices deployment problem as an integer programming problem under the constraint of the capacities of edge servers, which is proven as an NP-hard problem. To tackle this problem, we first configure replicas for each microservice to extend the application's DAG structure according to service requests and integrate all microservice replicas into a single DAG called Large-DAG (LDAG) for cooperative deployment. Obviously, the container replicas deployment for LDAG is also an NP-hard. To solve it, we convert the original inter-server communication minimization problem to an equivalent intra-server data dependency maximization problem. We propose a Deterministic Local Search-based Microservice Deployment (DLSMD) algorithm, that chooses a superior neighborhood solution iteratively to solve it and deploy each microservice replica into edge servers. Finally, we provide a theoretical analysis showing that the proposed DLSMD attains a guaranteed approximation ratio with low time complexity. Considering a more realistic edge environment where edge computing resources, communication resources, and container resource requirements are heterogeneous, we construct a fairer model that takes into account both the computation and communication time of the DAG workflows to minimize the total completion time, which is proved to be an NP-hard problem. To solve this problem, we propose a Heuristic Microservice Deployment (HMD) algorithm which is a variant of the DLSMD.

We conduct extensive simulations to evaluate the performance of proposed algorithms. The results show that DLSMD significantly outperforms other benchmark algorithms. Under the same system settings, our algorithm achieves the lowest communication overhead in a smaller iteration time. Compared with the Random algorithm, DLSMD can achieve up to 4× speedup in terms of reducing communication overhead. For a more general problem considering both computing and communication time, compared with the Random algorithm, our algorithm HMD outperforms baselines, achieving up to 3.12× speedup in terms of total completion time reduction.

The contributions of this paper are summarized as follows:

- We consider the effect of the data partition method for microservice deployment and model integer programming problems to minimize overall communication traffic and completion time.
- We propose the DLSMD and HMD algorithms to solve problems. We theoretically analyze the time complexity and approximation ratio of DLSMD to show the performance of DLSMD.
- We conduct extensive simulations to evaluate the performance of the algorithms. The results show that DLSMD and HMD significantly outperform other benchmark algorithms, achieving up to 4× and 3.12× speedups in terms of the reduction of inter-server traffic and total completion time.

2 Related Work

The MEC system is usually built on Kubernetes (K8s) to seamlessly integrate distributed computing resources on multiple edge servers to form edge server cluster[33, 34], and then provides services for end users by deploying containers on the K8s cluster[35]. However, due to the limited computing resources of edge servers, how to deploy services to meet the requirements of users has become a hot topic in recent years. The prior works can be divided into three categories according to the structure of the application: (1) service deployment problems for atomic applications, (2) service deployment problems for chained structured applications, and (3) service deployment problems for DAG structured applications.

Monolithic applications are considered to be an indivisible whole, and the deployment problem of such applications is considered to be modeled as an integer programming problem. In terms of latency in applications, Moubayed et al.[23] aiming at the edge scenario of Vehicle to Everything (V2X), modeled the cost of application service deployment into the sum of the delay of multiple stages of application execution, obtained an integer programming problem with minimal delay, and proposed a greedy algorithm to solve this problem. Considering the energy consumption generated by application services deployment, Li and Wang[24] proposed a model deployment strategy using particle swarm optimization to reduce the energy consumption generated by deployment strategy as far as possible under the constraint of meeting the maximum response delay of

the application. From the perspective of mobility, Ouyang et al.[25] considered the scenario in which services can be continuously migrated with user mobility and took computing delay, transmission delay, and service migration delay as minimization targets, and adopted a Markov approximation algorithm to obtain an approximate solution to the optimization problem. Qu et al.[36] developed a parameter-sharing model placement scheme for AI applications. Pan et al.[37] proposed a retention-aware container caching for SEC to make decisions on both distributing service requests to appropriate edge nodes and caching containers on these edge nodes. The above work can accelerate the service deployment of simple applications to some extent. However, as the structure of intelligent applications becomes more and more complex, simple service deployment is difficult to meet the needs of applications.

For the service deployment of chained applications, Yang et al.[26] modeled the service function chain into a chain structure containing multiple sub-services, considered the communication delay and calculation delay generated during the application execution, and proposed an approximate algorithm based on random rounding to obtain the approximate solution of this problem. Niu et al.[27] proposed a chain-oriented load balancing algorithm to determine how many instances of microservices to serve the request in chains. Jin et al.[38] investigated the virtual network function chains deployment problem with latency guarantees in MEC.

For the deployment of DAG structured application services, Bao et al.[28] modeled the execution process of the application into a DAG structure by splitting the service into multiple dependent sub-services, and minimized the delay consumption generated by DAG deployment through the multi-stage method based on graph pruning and genetic algorithm. Lin et al.[29] considered the deployment of the DNN model. In this study, the DNN model is modeled as DAG. The model deployment problem is divided into model segmentation and sub-model deployment problems, aiming at minimizing the execution time of tasks. A heuristic algorithm is designed to solve the problem. All of the above work is from the user's point of view, with the optimization goal of minimizing the execution time of the application under service deployment. However, from the perspective of app vendors, the communication cost between each sub-service should be considered. Chen et al.[30] and Huang and Shen[31] considered reducing the degree of network congestion in the cloud computing environment by minimizing the communication cost generated by the deployment of sub-services. However, the cloud computing environment has almost infinite computing resources, so it is difficult to apply to the edge environment with limited resources. Wang et al.[32] considered the problem of service deployment under resource constraints and used a heuristic algorithm to minimize transmission costs. However, this heuristic algorithm makes it difficult to obtain a performance guarantee in a dynamic edge environment. Fu et al.[18] considered the microservice deployment problem of the DAG structure and adopted reinforcement learning to minimize the communication cost of sub-service deployment at different edge server nodes. However, existing work does not consider how to deploy multiple DAG-structured applications in an edge computing environment, and each microservice may have multiple service requests.

3 System Model and Problem Formulation

In this section, we first introduce the system model and then we present the problem formulation of this paper.

3.1 System model

We consider an MEC system that consists of a set $\mathcal{N} = \{1, 2, \ldots, N\}$ of Edge Servers (ESs). These edge servers are interconnected with each other by a public network connection. To ensure the quality of service, the number of container replicas deployed in ES n cannot be more than edge server capacities C_n, $n \in \mathcal{N}$. Microservices on the same edge server equal share resources by default.

The microservice structure of any application can be described as a DAG. We consider there are a set $\mathcal{G} = \{G_1, G_2, \ldots\}$ of application microservices that need to be deployed in the edge. We construct an application $G_i = (V_i, E_i)$, where V_i denotes the set of microservices of G_i and E_i denotes the communications between microservices. Let vertex v_{ij} denotes the j-th microservice in application G_i and $e_{ij,ij'}$ denotes the edge between the j-th and the j'-th microservices in application G_i. And the weight of edge $e_{ij,ij'}$ is $w(e_{ij,ij'})$, which denotes the communication traffic between microservices.

We consider every microservice may have multiple service requests. The number of requests for

microservices is considered as the weight of the vertex v_{ij}, which is denoted as $|l(v_{ij})|$. To enable the parallel execution of sub-tasks, we need to deploy a container replica for each service request. Thus, let v_{ijk} denote the k-th container replica of microservice v_{ij}. We suppose that the communication traffic between any two microservices will be equally divided by the number of new links generated by the multiple container replicas. So we have

$$w(e_{ij_1k_1, ij_2k_2}) = \frac{w(e_{ij_1, ij_2})}{l(v_{ij_1})l(v_{ij_2})},$$
$$k_1 \in [1, l(v_{ij_1})],\ k_2 \in [1, l(v_{ij_2})] \tag{1}$$

It is worth noting that this relationship of Eq. (1) can be alternated be other models according to application characters.

3.2 Cost model

According above formulations, we define the variable $I^n_{ijk} \in \{0, 1\}$, $n \in \mathcal{N}$ to indicate whether the container replica v_{ijk} is deployed in edge server n or not. Therefore, the communication traffic between any two container replicas can be expressed as follows:

$$T_{ij_1k_1, ij_2k_2} = w(e_{ij_1k_1, ij_2k_2}) I^n_{ij_1k_1} I^{n'}_{ij_2k_2} f(n, n') \tag{2}$$

where $f(n, n') \in \{0, 1\}$ is a bool function. It returns 0 if $n = n'$ and 1 otherwise, indicating whether n equals n' or not. Therefore, Eq. (2) refers to that if microservices $v_{ij_1k_1}$ and $v_{ij_2k_2}$ are deployed onto the same edge server node, the communication overhead equals 0 due to the data integration between microservices can be transmitted through global memory instead of the network. Otherwise, the communication traffic between microservices $v_{ij_1k_1}$ and $v_{ij_2k_2}$ is $w(e_{ij_1k_1, ij_2k_2})$.

3.3 Problem formulation

Thus, the overall communication traffic of all microservices in the system can be expressed as follows:

$$T_{\text{total}} = \sum_{i=1}^{|\mathcal{G}|} \sum_{j_1=1}^{|V_i|} \sum_{j_2=1}^{|V_i|} \sum_{k_1=1}^{|l(v_{ij_1})|} \sum_{k_2=1}^{|l(v_{ij_2})|} T_{ij_1k_1, ij_2k_2} \tag{3}$$

Our goal is to minimize the overall communication traffic of the edge computing system, which can be formulated as follows:

$$\text{(P1)}: \min_{\mathcal{I}} T_{\text{total}} \tag{4}$$

s.t.,

$$\text{C1: } I^n_{ijk} \in \{0, 1\}, \forall i \in \{1, 2, \ldots, |\mathcal{G}|\},\ j \in \{1, 2, \ldots, |V_i|\},\ n \in \mathcal{N} \tag{5}$$

$$\text{C2: } \sum_{n \in \mathcal{N}} I^n_{ijk} = 1,\ \forall i \in \{1, 2, \ldots, |\mathcal{G}|\},\ j \in \{1, 2, \ldots, |V_i|\} \tag{6}$$

$$\text{C3: } \sum_{i \in |\mathcal{G}|} \sum_{j \in |V_i|} I^n_{ijk} \leqslant C_n,\ \forall n \in \mathcal{N} \tag{7}$$

where $\mathcal{I} = \{I^n_{ijk} | i \in \{1, 2, \ldots, |\mathcal{G}|\},\ j \in \{1, 2, \ldots, |V_i|\},\ n \in \mathcal{N}\}$. Constraint C1 expresses the container replica deployment decision. Constraint C2 claims that every container replica deploys on one and only one edge server node. Constraint C3 means that the number of containers deployed on the edge server node n cannot be more than the capacity constraint. We can find that Problem P1 is an integer programming problem in which the variable I^n_{ijk} is an integer. Since every container replica can be deployed onto any edge server, the possible solutions space of Problem (P1) is $\left(\sum_{i=1}^{|\mathcal{G}|} \sum_{j=1}^{|V_i|} \sum_{k=1}^{|l(v_{ij})|}\right)^{|\mathcal{N}|}$ which is exponential. The Problem P1 can be proven to be an NP-hard problem by reducing it from the min-k-cut problem. The min-k-cut problem is the problem of partitioning vertices of a given graph or hypergraph into k subsets, such that the total weight of edges or hyperedges crossing different subsets is minimized. In Problem (P1), we consider a special case where all edge servers have sufficient resources to place all container replicas and the number of applications is 1. This is a typical min-k-cut problem, which has been proved as an NP-hard problem when $k \geqslant 3$[39]. Hence, Problem (P1) is NP-hard.

4 Application Integration and Microservice Deployment

In this section, we integrate all the application services into one DAG and propose a method to deploy microservices to edge servers to minimize overall communication overhead.

4.1 Application integration method

App vendors need to deploy microservices for multiple applications simultaneously. Collaborative microservices deployment is non-trivial, because separate deployment for each application results in sub-optimal performance. What is more, as mentioned above, every microservice may have multiple service requests. To achieve parallel execution acceleration for end users, app vendors need to deploy multiple

microservice replicas in advance. Therefore, we convert all microservices replicas of all applications into one DAG called LDAG for collaborative deployment, minimizing communication overhead. In the following, we propose Algorithm 1 to integrate all application microservices into LDAG.

Algorithm 1 consists of two phases: (1) We construct new DAGs by extending every DAGs by the number of requests each microservice (Lines 1–12); (2) We integrate all extended application DAGs into LDAG (Lines 13–17). In the first phase, we add vertices to the set V_i' according to the number of requests of each microservice. Then, we add edge $e_{ij_1k_1, ij_2k_2}$ for any replica of microservice v_{ij_1} and v_{ij_2} to build E_i', if there is an edge e_{ij_1, ij_2} in original G_i. We set the weight for edge $e_{ij_1k_1, ij_2k_2}$ according Eq. (1). Finally, we combine V_i' and E_i' to construct $G_i' = (V_i', E_i')$. In the second phase, we integrate all G_i' into LDAG. We first add "beigin" and "end" vertices v_{begin} and v_{end} into LDAG, respectively. For each G_i', we put all vertices and edges into LDAG, and add edges $(v_{\text{begin}}, v_{i0})$ and $(v_{i(-1)}, v_{\text{end}})$, where v_{i0} denotes the first vertex of G_i', (i.e., vertex with in-degree 0) and $v_{i(-1)}$ denotes the last vertex of G_i' (i.e., vertex with out-degree 0). And then, we set the weights of $(v_{\text{begin}}, v_{i0})$ and $(v_{i(-1)}, v_{\text{end}})$ as 0.

Algorithm 1 Application integration algorithm

Input: $\mathcal{G}$

Output: LDAG

1: **for** each $G_i \in \mathcal{G}$ **do**
2: **for** each $v_{ij} \in V_i$ **do**
3: **for** $k \in [1, |l(v_{ij})|]$ **do**
4: Add vertex v_{ijk} to V_i';
5: **end for**
6: **end for**
7: **for** each $e_{ij_1, ij_2} \in E_i$ **do**
8: Add edge $e_{ij_1k_1, ij_2k_2}$ to E_i', $\forall k_1 \in [1, |l(v_{ij_1})|]$, $\forall k_2 \in [1, |l(v_{ij_2})|]$;
9: Compute the weight of edge $e_{ij_1k_1, ij_2k_2}$ according to Eq. (1), $\forall k_1 \in [1, |l(v_{ij_1})|]$, $\forall k_2 \in [1, |l(v_{ij_2})|]$;
10: **end for**
11: Construct $G_i' = (V_i', E_i')$ and add G_i' to $\mathcal{G}'$;
12: **end for**
13: Add two vertices v_{begin} and v_{end} to LDAG;
14: **for** each G_i' in $\mathcal{G}'$ **do**
15: Add G_i' to LDAG;
16: Add edges $(v_{\text{begin}}, v_{i0})$ and $(v_{i(-1)}, v_{\text{end}})$ to LDAG;
17: Set weights $(v_{\text{begin}}, v_{i0}) \leftarrow 0$ and $(v_{i(-1)}, v_{\text{end}}) \leftarrow 0$;
18: **end for**
19: **return** LDAG

4.2 Microservice deployment based on deterministic local search

According to the above application integration, the essential problem is to partition LDAG into $|\mathcal{N}|$ parts and distribute them to edge servers to minimize the sum of weights of edges across all pairs of edge server nodes, which can be modeled as a min-k-cut problem intuitively. Inspired by Refs. [39, 40], we propose a DLSMD.

In the following, before proposing a microservice deployment method, we first redefine notations for LDAG for ease of expression. We define LDAG = (V, E), where V denotes the set of all microservice replicas and E denotes the set of communication overhead between microservice replicas. We denote V_n, $n \in \mathcal{N}$, the vertex set which is deployed onto the edge server n.

Let edge $e \in E$ connect vertices $u, v \in V$, and let $d(u, v)$ denote the weight on edge $e_{u,v}$, i.e.,

$$d(u, v) = w(e_{u,v}) \tag{8}$$

Let $d(u, V_m)$ be the sum of the edges from vertex u to the vertices in the subset V_m, i.e.,

$$d(u, V_m) = \sum_{v \in V_m, v \neq u} d(u, v) \tag{9}$$

Let $d(V_n, V_n)$ denote the sum of weights of edges with both ends in V_n. Let $d(V_n, V_m)$ denote the sum of weights of edges that connect a vertex in V_n to a vertex in V_m.

For ease of understanding, the meanings of $d(u, V_m)$, $d(V_m, V_m)$, and $d(V_m, V_n)$ are shown in Fig. 2 by red lines.

According to the above definitions, we have the following relationships:

$$2d(V_m, V_m) = \sum_{u \in V_m} d(u, V_m) \tag{10}$$

$$d(V_m, V_n) = \sum_{v \in V_n} \sum_{u \in V_m} d(v, u) \tag{11}$$

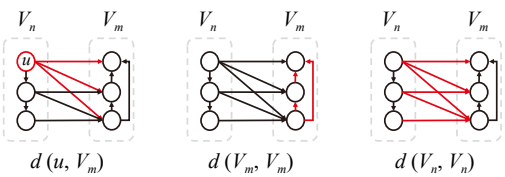


Fig. 2 Illustration of $d(u, V_m)$, $d(V_m, V_m)$, and $d(V_m, V_n)$.

Furthermore, we denote d_{cut} the sum of weights of the cut edges, and d_{noncut} the sum of weights of edges that do not belong to d_{cut}. According to the above definitions, we can express d_{cut} and d_{noncut} as follows:

$$d_{\text{cut}} = \sum_{i=1}^{|\mathcal{N}|} \sum_{j=i+1}^{|\mathcal{N}|} d(V_i, V_j) \tag{12}$$

$$d_{\text{noncut}} = \sum_{i=1}^{|\mathcal{N}|} d(V_i, V_i) \tag{13}$$

The sum of d_{cut} and d_{noncut} is the sum of weights of all edges in E, which denotes as d_{sum},

$$d_{\text{sum}} = d_{\text{cut}} + d_{\text{noncut}} \tag{14}$$

Recall that, the goal of this paper is to minimize the sum of weights of edges across all pairs of edge server nodes, i.e., d_{cut}. According to Eq. (14), we can turn the original problem into maximizing d_{noncut}, i.e.,

$$\text{(P2)}: \max_{\mathcal{I}} d_{\text{noncut}} \tag{15}$$

$$\text{s.t., C1}: I_i^n \in \{0, 1\},\ \forall i \in \{1, 2, \ldots, |V|\},\ n \in \mathcal{N} \tag{16}$$

$$\text{C2}: \sum_{n \in \mathcal{N}} I_i^n = 1,\ \forall i \in \{1, 2, \ldots, |\mathcal{G}|\} \tag{17}$$

$$\text{C3}: \sum_{i=1}^{|V|} I_i^n \leqslant C_n,\ \forall n \in \mathcal{N} \tag{18}$$

where $\mathcal{I} = \{I_i^n | i \in \{1, 2, \ldots, |V|\}, n \in \mathcal{N}\}$ and I_i^n indicates whether the microservice replica v_i is deployed in edge server n or not. The meaning of constraints is similar to that of Problem (P1), so we do not repeat it.

In the following, we propose a local search-based algorithm[41] to solve Problem (P2). The local search algorithm starts from an initial state and continuously generates the neighborhood solution of the current solution state by executing the neighborhood action designed based on the corresponding problem. It determines whether the neighborhood solution is superior to the current solution through the strategy function designed by the definition of the objective function of the problem, and iteratively updates the current solution. The above actions are repeated until the current solution is locally optimal, which is the final output of the algorithm.

Motivated by the above key idea, we propose a local search-based microservice deployment algorithm. The pseudo-code is presented in Algorithm 2. Firstly, all microservice vertices are randomly placed on the edge server for initialization (Line 2). The key step of algorithm design is described as follows:

Key step (Lines 3–12): If there are vertices $u \in V_m$, $v \in V_n$, $n \neq m$, such that

$$d(u, V_m) + d(v, V_n) < d(u, V_n) + d(v, V_m) - 2d(u, v) \tag{19}$$

If such a pair of vertices exists, reassign vertex u to V_n, and vertex v to V_m. Algorithm 2 terminates when there does not exist any pair of vertices satisfying Formula (19).

The final output of Algorithm 2 is the solution of Problem (P2).

5 Theoretical Analysis

In this section, we first prove the time complexity of Algorithm 1. Then, we introduce the convergence, time complexity, and the worst-case bound of Algorithm 2.

Theorem 1 The time complexity of Algorithm 1 is $O(G_{\max}|\mathcal{L}_v|(V_{\max} + E_{\max}|\mathcal{L}_v|))$, where $|\mathcal{L}_v| = \max\{l(v_{ij}), \forall i \in \{1, 2, \ldots, |\mathcal{G}|\},\ j \in \{1, 2, \ldots, |V_i|\}\}$, $G_{\max} = \max\{G_i, \forall i \in \{1, 2, \ldots, |\mathcal{G}|\}\}$, $V_{\max} = \max\{V_i, \forall i \in \{1, 2, \ldots, |\mathcal{G}|\}\}$, and $E_{\max} = \max\{E_i, \forall i \in \{1, 2, \ldots, |\mathcal{G}|\}\}$.

Proof Now, let us analyze the time complexity of Algorithm 1. Lines 2–4 involve adding each microservice $v_{ij} \in V_i$ to V_i', yielding a time complexity of $O(V_{\max}|\mathcal{L}_v|)$. From Lines 7–10, incorporating each edge $e_{ij_1, ij_2} \in E_i$ into E_i' entails a time complexity of $O(E_{\max}|\mathcal{L}_v|^2)$. Constructing G_i' in Line 11 is an $O(1)$

Algorithm 2 Pseudocode of DLSMD

Input: $\mathcal{G}$

Output: V_n, $n \in \mathcal{N}$

```
1: Integrate G into LDAG by calling Algorithm 1;
2: Initialization: Place V into N randomly;
3: repeat
4:    ind ← 1; // ind denotes an indication
5:    for u ∈ V_m, m ∈ N do
6:       for v ∈ V_n, n ∈ N, n ≠ m do
7:          if d(u,V_m) + d(v,V_n) < d(u,V_n) + d(v,V_m) − 2d(u,v)
            then
8:             Reassign vertex u to V_n, vertex v to V_m;
9:             ind ← 0;
10:         end if
11:      end for
12:   end for
13: until ind = 1;
14: return V_n, n ∈ N
```

operation. Thus, the time complexity for Lines 1−12 amounts to $O(G_{\max}|\mathcal{L}_v|(V_{\max}+E_{\max}|\mathcal{L}_v|))$, considering the contributions from the aforementioned steps.

Moving on to Line 13, adding the start and end vertices to LDAG has a time complexity of $O(1)$. For the block of code from Lines 14−18, the algorithm incorporates each G_i' into $\mathcal{G}'$, with a time complexity of $O(|\mathcal{G}'|)$. Given the context of the algorithm, we can get that $|\mathcal{G}|=|\mathcal{G}'|$. Consequently, we can deduce that the time complexity of Algorithm 1 is $O(G_{\max}|\mathcal{L}_v|(V_{\max}+E_{\max}|\mathcal{L}_v|))$. ■

We denote ALG the objective value of an intermediate solution produced by Algorithm 2, which represents the sum of weights of edges in V_n, $n\in\mathcal{N}$ in the intermediate state, i.e.,

$$\mathrm{ALG}=\sum_{n\in\mathcal{N}} d(V_n, V_n) \tag{20}$$

Lemma 1 Let ALG′ denote the objective value of current solution and ALG″ denote the objective value of the solution after any iterative step of Algorithm 2. We have

$$\mathrm{ALG}''-\mathrm{ALG}'\geqslant 1 \tag{21}$$

Proof In this paper, the weight of edges denotes the communication overhead. Without loss of generality, the minimum unit of communication overhead is assumed to be 1. According to Formula (19), after any iterative step, the ALG′ will decrease, which proves Lemma 1. ■

Theorem 2 Algorithm 2 terminates in a finite number of steps. The time complexity of Algorithm 2 is $O(|V|^2 d_{\mathrm{sum}}+G_{\max}|\mathcal{L}_v|(V_{\max}+E_{\max}|\mathcal{L}_v|))$.

Proof According to Theorem 1, Line 1 exhibits a time complexity of $O(G_{\max}|\mathcal{L}_v|(V_{\max}+E_{\max}|\mathcal{L}_v|))$. Moving on to Line 2, vertexes are randomly placed onto edge servers, resulting in a complexity of $O(|V|)$. According to Lemma 1, after any iterative step of Algorithm 2, ALG increases by at least 1. Obviously, ALG is always less than d_{sum}. Therefore, Algorithm 2 terminates in a finite number of steps. The maximum number of iterations is d_{sum}. For every iterative step, it spends at most $O(|V|^2)$ to search a case which satisfies Formula (19). Therefore, the time complexity of Algorithm 2 is $O(|V|^2 d_{\mathrm{sum}}+G_{\max}|\mathcal{L}_v|(V_{\max}+E_{\max}|\mathcal{L}_v|))$, which proves Theorem 2. ■

In the following, we will analyze the performance of Algorithm 2 by deriving its Approximation Ratio (AR) ρ, which is the ratio of the value obtained by the Algorithm 2 to the optimal value. AR is defined as follows:

$$\rho=\frac{\mathrm{ALG}^*}{\mathrm{OPT}} \tag{22}$$

where ALG^* denotes the value of final solution of Algorithm 2 and OPT denotes the value of the optimal solution.

Theorem 3 The approximation ratio of Algorithm 2 is $\dfrac{t-1}{s(|\mathcal{N}|-1)+t-1}$, where $t=\min_{m\in\mathcal{N}}|V_m|$ and $s=\max_{m\in\mathcal{N}}|V_m|$.

Proof According to Algorithm 2, when the algorithm terminates, all pairs of vertices ($u\in V_m$ and $v\in V_n$) do not satisfy the Formula (19). That is, all pairs of (u, v) meet the following inequation:

$$d(u, V_m)+d(v, V_n)\geqslant d(u, V_n)+d(v, V_m)-2d(u, v),\quad \forall u\in V_m, v\in V_n \tag{23}$$

We sum both sides of Formula (23) for $u\in V_m$ and get

$$\sum_{u\in V_m}(d(u, V_m)+d(v, V_n))\geqslant \sum_{u\in V_m}(d(u, V_n)+d(v, V_m)-2d(u, v)) \tag{24}$$

Since $d(v, V_n)$ and u are independent, thus $\sum_{u\in V_m} d(v, V_n)=|V_m|d(v, V_n)$. Therefore, we can rewrite Formula (24) as follows:

$$\sum_{u\in V_m} d(u, V_m)+|V_m|d(v, V_n)\geqslant \sum_{u\in V_m} d(u, V_n)+|V_m|d(v, V_m)-\sum_{u\in V_m}2d(u, v) \tag{25}$$

Similarly, we sum both side of the Formula (25) for all $v\in V_n$ and get

$$|V_n|\sum_{u\in V_m} d(u, V_m)+|V_m|\sum_{v\in V_n} d(v, V_n)\geqslant |V_n|\sum_{u\in V_m} d(u, V_n)+|V_m|\sum_{v\in V_n} d(v, V_m)-2\sum_{v\in V_n}\sum_{u\in V_m} d(v, u) \tag{26}$$

We substitute Eqs. (10) and (11) into Formula (26) to get

$$2|V_n|d(V_m, V_m)+2|V_m|d(V_n, V_n)\geqslant |V_n|d(V_m, V_n)+|V_m|d(V_n, V_m)-2d(V_m, V_n) \tag{27}$$

Due to $d(V_m, V_n)=d(V_n, V_m)$, we can get

$$2|V_n|d(V_m, V_m)+2|V_m|d(V_n, V_n)\geqslant (|V_m|+|V_n|-2)d(V_m, V_n) \tag{28}$$

Then, we sum both sides of Formula (28) for

$n, m \in \mathcal{N}$ and get

$$2\sum_{m\in\mathcal{N}}\sum_{n\in\mathcal{N},\, m\neq n}(|V_n|d(V_m, V_m)+|V_m|d(V_n, V_n)) \geqslant \sum_{m\in\mathcal{N}}\sum_{n\in\mathcal{N},\, m\neq n}((|V_m|+|V_n|-2)d(V_n, V_m)) \tag{29}$$

We define $s=\max_{m\in\mathcal{N}}|V_m|$ to scale the left side of Formula (29). Thus, we have

$$2s\sum_{m\in\mathcal{N}}\sum_{n\in\mathcal{N},\, m\neq n}(d(V_m, V_m)+d(V_n, V_n)) \geqslant 2\sum_{m\in\mathcal{N}}\sum_{n\in\mathcal{N},\, m\neq n}(|V_n|d(V_m, V_m)+|V_m|d(V_n, V_n)) \tag{30}$$

Further, the left side of Formula (30) can be rewritten as follows:

$$2s\sum_{m\in\mathcal{N}}\sum_{n\in\mathcal{N},\, m\neq n}(d(V_m, V_m)+d(V_n, V_n)) = 2s(|\mathcal{N}|-1)\left(\sum_{m\in\mathcal{N}}d(V_m, V_m)+\sum_{n\in\mathcal{N}}d(V_n, V_n)\right) \tag{31}$$

Because $\sum_{m\in\mathcal{N}}d(V_m, V_m)=\sum_{n\in\mathcal{N}}d(V_n, V_n)$ and $\mathrm{ALG}^*=\sum_{n\in\mathcal{N}}d(V_n, V_n)$, we have

$$2s(|\mathcal{N}|-1)\left(\sum_{m\in\mathcal{N}}d(V_m, V_m)+\sum_{n\in\mathcal{N}}d(V_n, V_n)\right) = 4s(|\mathcal{N}|-1)\mathrm{ALG}^* \tag{32}$$

Similarly, we define $t=\min_{m\in\mathcal{N}}V_m$ to scale the right side of Formula (29),

$$\sum_{m\in\mathcal{N}}\sum_{n\in\mathcal{N},\, m\neq n}((|V_m|+|V_n|-2)d(V_N, V_m)) \geqslant \sum_{m\in\mathcal{N}}\sum_{n\in\mathcal{N},\, m\neq n}((2t-2)d(V_m, V_n)) \tag{33}$$

According to the Eq. (12), we have

$$\sum_{m\in\mathcal{N}}\sum_{n\in\mathcal{N},\, m\neq n}((2t-2)d(V_m, V_n)) = 4(t-1)d_{\mathrm{cut}} \tag{34}$$

According to the above analysis, we have

$$4s(|\mathcal{N}|-1)\mathrm{ALG}^* \geqslant 2\sum_{m\in\mathcal{N}}\sum_{n\in\mathcal{N},\, m\neq n}(|V_n|d(V_m, V_m)+|V_m|d(V_n, V_n)) \geqslant \sum_{m\in\mathcal{N}}\sum_{n\in\mathcal{N},\, m\neq n}((|V_m|+|V_n|-2)d(V_n, V_m)) \geqslant 4(t-1)d_{\mathrm{cut}} \tag{35}$$

Therefore, we have

$$4s(|\mathcal{N}|-1)\mathrm{ALG}^* \geqslant 4(t-1)d_{\mathrm{cut}} \tag{36}$$

$$d_{\mathrm{cut}} \leqslant \frac{s(|\mathcal{N}|-1)}{t-1}\mathrm{ALG}^* \tag{37}$$

Because OPT must less than $d_{\mathrm{sum}}=d_{\mathrm{cut}}+\mathrm{ALG}^*$, therefore,

$$\mathrm{OPT} \leqslant d_{\mathrm{cut}}+\mathrm{ALG}^* \leqslant d_{\mathrm{cut}}+ALG^* \leqslant \frac{s(|\mathcal{N}|-1)}{t-1}\mathrm{ALG}^*+\mathrm{ALG}^* = \frac{s(|\mathcal{N}|-1)+t-1}{t-1}\mathrm{ALG}^* \tag{38}$$

Finally, we have

$$\rho=\frac{\mathrm{ALG}^*}{\mathrm{OPT}} \geqslant \frac{t-1}{s(|\mathcal{N}|-1)+t-1} \tag{39}$$

which proves the theorem. ■

According to Theorem 2, we have a corollary as follows.

Corollary 1 For better performance, replicas should be distributed evenly across edge servers, i.e., let s/t be as small as possible.

Proof Let us divide the numerator and the denominator of Formula (39) by $t-1$, we can get

$$\rho=\frac{\mathrm{ALG}^*}{\mathrm{OPT}} \geqslant \frac{1}{s\frac{(|\mathcal{N}|-1)}{(t-1)}+1} \tag{40}$$

Therefore, the AR is determined by $s/(t-1)$. When $s/(t-1)$ decreases, the AR increases, which proves the corollary. ■

According to Corollary 1, microservice replicas should be distributed across edges as evenly as possible.

6 Microservice Deployment with Computing & Communication

In a more general context of microservice placement problems, our focus extends beyond merely minimizing communication overhead to deeply considering the impact of deployment locations on the completion time of microservices. This encompasses both computation time and transmission time. In practical scenarios, edge servers exhibit heterogeneity in both their computing capabilities and inter-server transmission capacities. Consequently, the deployment of microservices becomes a pivotal factor in determining both computation and transmission time. To ensure fairness and optimize performance, we have constructed a more versatile model for microservice deployment. We denote the computing capacity of edge server $n \in \mathcal{N}$ as C_n, the computing demand of a microservice $u \in V$ as c_u, and the set of the predecessor microservices of microservice u as $p(u)$. To capture

the heterogeneity in communication between edge servers, we use B_{mn} to indicate the bandwidth size between edge servers m and n, $m, n \in V$. In the microservice deployment problem, we need to decide on which edge server each microservice $u \in V$ should be deployed. Therefore, we use the indicator variable $x_{un} \in \{0, 1\}$ to denote whether microservice u is deployed on edge server n.

6.1 Problem formulation

Due to the heterogeneity in the types and capacities of computing resources among edge servers, executing microservice $u \in V$ on different edge servers may result in varied computing time. We use q_{un} to denote the computing time of microservice $u \in V$ on edge server $n \in \mathcal{N}$. Thus, the computing time of microservice $u \in V$ can be expressed as

$$\tau_{\text{comp}}(u) = \sum_{n \in \mathcal{N}} x_{un} q_{un},\ \forall u \in V \tag{41}$$

If there is an edge between microservice v and microservice u in the LDAG, then transmitting dependent data between them will incur a transmission time. The transmission time between microservice $v \in V$ and $u \in V$ can be

$$\tau_{\text{tran}}(v, u) = \sum_{m \in V} \sum_{n \in V} \frac{x_{vm} x_{un} w(e_{v,u}) f(m, n)}{B_{mn}} \tag{42}$$

where $v \neq u$. If m equals n, $f(m, n)$ equals 0, indicating that microservice $v \in V$ and $u \in V$ are deployed on the same edge server, with a communication time of 0. Therefore, the microservice u's completion time $\tau_{\text{end}}(u)$ can be expressed as

$$\tau_{\text{end}}(u) = \tau_{\text{sta}}(u) + \tau_{\text{comp}}(u),\ \forall u \in V \tag{43}$$

where $\tau_{\text{sta}}(u)$ is the start time of microservice $u \in V$, which can be expressed as

$$\tau_{\text{sta}}(u) = \max\{\tau_{\text{end}}(v) + \tau_{\text{tran}}(v, u),\ \forall v \in p(u)\} \tag{44}$$

The completion time $\tau_{\max}$ of LDAG is equivalent to the maximum completion time of all microservices, we have

$$\tau_{\max} = \max\{\tau_{\text{end}}(u),\ \forall u \in V\} \tag{45}$$

Our optimization goal is to deploy microservices in a heterogeneous edge environment to minimize the completion time of a batch of microservice applications, that is, to minimize the completion time of LDAG. Therefore, a more general microservice deployment problem can be formulated as

$$\text{(P3)}: \min_{\boldsymbol{x}} \tau_{\max} \tag{46}$$

$$\text{s.t., C1}: \sum_{n \in \mathcal{N}} x_{un} = 1 \tag{47}$$

$$\text{C2}: \sum_{u \in V} x_{un} c_n \leqslant C_n \tag{48}$$

$$\text{C3}: x_{un} \in \{0, 1\} \tag{49}$$

where $\boldsymbol{x} = \{x_{un} | u \in V, n \in \mathcal{N}\}$. Constraint C1 ensures that each microservice can be deployed on only one edge server. Constraint C2 ensures that the sum of computing resources required by microservices deployed on each edge server cannot exceed its computing capability constraint. Constraint C3 indicates that x_{un} is an integer.

6.2 Complexity analysis

Problem (P3) is an NP-hard problem. We provide a brief proof of this. We consider a special case of Problem (P3), denoted as (P3-C), where communication among microservices is not taken into account. Therefore, (P3-C) can be viewed as a variant of the classic bin packing problem, where each microservice is considered an item and each edge server is treated as a bin. Since the classic bin packing problem is an NP-hard problem, Problem (P3), as a more complex variant, is also NP-hard.

6.3 Heuristic algorithm

Considering the NP-hardness of Problem (P3), finding an optimal solution within polynomial time is impossible. To address this issue, we propose a heuristic microservice deployment algorithm.

We use $\tau_{\max}(u, n, v, m)$ to denote the completion time of the LDAG when microservices u and v are deployed on edge servers n and m, respectively, while the deployment servers of other microservices remain unchanged. The details of the algorithm are presented in Algorithm 3. Firstly, we invoke Algorithm 1 to integrate the microservice applications $\mathcal{G}$ into an LDAG (Line 1). We initialize the deployment of all microservices randomly (Line 2). Then, for any two microservices u and v deployed on different edge servers n and m, $n \neq m$, if swapping microservices u and v can reduce the completion time $T_{\max}$ of the LDAG, that is $\tau_{\max}(u, m, v, n) < \tau_{\max}(u, n, v, m)$, we swap the deployment locations of u and v. This process continues until no pair of microservices can be swapped to further decrease the LDAG's completion time (Lines 3 – 13). Finally, we set the x_{un} values based

Algorithm 3 HMD algorithm

Input: $\mathcal{G}$

Output: x_{un}, $\forall n \in \mathcal{N}$, $u \in V$

1: Integrate $\mathcal{G}$ into LDAG by calling Algorithm 1;
2: **Initialization:** Place V into $\mathcal{N}$ greedily according to the order of vertexes and obtain the set V_n of microservices deployed on each edge server n;
3: **repeat**
4: ind $\leftarrow$ 1;
5: **for** $u \in V_n, n \in \mathcal{N}$ **do**
6: **for** $v \in V_m, m \in \mathcal{N}, n \neq m$ **do**
7: **if** $\tau_{\max}(u, m, v, n) < \tau_{\max}(u, n, v, m)$ **then**
8: Reassign vertex u to V_m and vertex v to V_n;
9: ind $\leftarrow$ 0;
10: **end if**
11: **end for**
12: **end for**
13: **until** ind = 0;
14: **for** $u \in V$ **do**
15: **if** $u \in V_n$, $\forall n \in \mathcal{N}$ **then**
16: set $x_{un} = 1$;
17: **end if**
18: **end for**
19: **return** $\{x_{un}\}$

on the deployment location of each microservice after all swaps (Lines 14 – 18).

6.4 Time complexity of HMD

Theorem 4 The time complexity of Algorithm 3 is $O\left(\frac{\tau'}{\epsilon}|V|^2 + G_{\max}|\mathcal{L}_v|(V_{\max} + E_{\max}|\mathcal{L}_v|)\right)$, where ϵ is the smallest unit of completion time and τ' is the execution time for LDAG using the lowest computing power and network bandwidth, representing the worst-case for LDAG's completion time.

Proof Let us delve into the time complexity analysis of Algorithm 3. According to Theorem 1, Line 1 exhibits a time complexity of $O(G_{\max}|\mathcal{L}_v|(V_{\max} + E_{\max}|\mathcal{L}_v|))$. Moving on to Line 2, vertexes are greedily placed onto edge servers, resulting in a complexity of $O(|V|)$. Drawing parallels with Theorem 2, we introduce ϵ as the smallest unit of completion time and τ' as the execution time for LDAG using the lowest computing power and network bandwidth, representing the worst-case for LDAG's completion time.

Within the loop from Lines 5 to 12, each iteration guarantees a reduction in LDAG's completion time by at least ϵ, contributing to a time complexity of $O(|V|^2)$. As for Lines 3 to 13, it involves swapping any pair of microservices that can decrease LDAG's completion time, the process that repeats at most τ/ϵ times. Hence, the complexity of this process can be denoted as $O(\tau'|V|^2/\epsilon)$. Lastly, from Lines 14 to 18, based on the iteration results, x_{uv} is set to either 0 or 1, yielding a complexity of $O(|V||\mathcal{N}|)$. Thus, the time complexity of Algorithm 3 is $O\left(\frac{\tau'}{\epsilon}|V|^2 + G_{\max}|\mathcal{L}_v|(V_{\max} + E_{\max}|\mathcal{L}_v|)\right)$.

7 Simulation Result

We next demonstrate the performance of the proposed algorithm via numerical simulations. We first describe the simulation settings. Then the benchmark algorithms and numerical results are presented.

7.1 Simulation setup

Our simulations run on a server with the operation system as Ubuntu 18.04 LTS, CPU as 8 cores Intel i7-8700CPU of 3.20 GHz, and memory as 8 GB RAM. To simulate a real MEC environment, we refer to the literature[42] and give some parameter settings related to the edge server nodes and applications. The edge computing cluster system contains 10–30 edge servers. The capacity of each edge server ranges from 30 to 50. The number of applications ranges from 1 to 10, and each application contains 10–50 microservices. Each microservice contains 10–50 service requests. The data size between microservices follows the uniform distribution of [20 KB, 50 KB]. To ensure that all microservice replicas can be deployed into the edge computing cluster, we assume that the total capacity of the edge computing cluster is greater than the total number of microservice replicas.

To verify the effectiveness of the algorithm, we synthesize the DAG workflow dataset used in this paper based on DAG Generator[43, 44] and Alibaba data trace[45, 46]. Similar to Refs. [43, 44], DAG generator generates several DAG data to represent the list of applications that contain microservices to be deployed. The DAG generator randomly generates DAGs with different topologies by controlling the {num, fat, density} triplet parameters, where "num" denotes the number of vertices of DAGs (i.e., the number of microservices), the "fat" is used to control the height of the DAG structure and the width in each layer, and "density" is used to control the density of edges of two adjacent layers in DAG. To facilitate understanding of the triplet parameters, Fig. 3 shows the DAGs under different parameters. Based on the

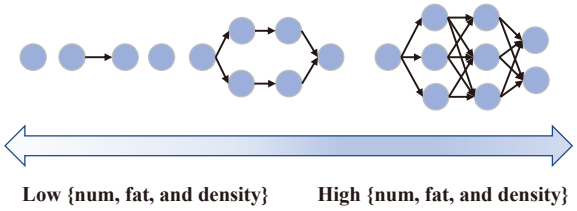


Fig. 3 Illustration of DAGs with different triplet parameters.

DAG generator, we use the real data Alibaba data trace to verify the effectiveness of the proposed algorithm in real situations. Alibaba data trace is generated by the Alibaba Cloud platform, which is widely used in the simulation experiment for DAG applications in an edge computing environment[45, 46]. Analysis of this dataset reveals that 90% of the subtasks in DAGs have a quantity of less than 50. Based on this characteristic, when generating DAGs using the DAG generator, the number of subtasks is also controlled within the range of [10, 50].

For the general case, considering both computing and communication time, we randomly set the computing time within the range of [0.0001 s, 1.1432 s][47], which represents the average execution delay for three types of microservices, i.e., data pre-processing, machine learning inference with a DNN, and results feedback to end users, performed on four Raspberry Pi 4 devices and two NVIDIA Jetson Xavier NX devices. According to the minimal and maximal memory consumption of these three microservices, i.e., 200 MB and 1.5 GB, similar to the work[47], we randomly set the range of the resource requirement of each microservice as [0.2 GB, 1.5 GB]. The total memory of each Raspberry Pi 4 and each NVIDIA Jetson Xavier NX is 8 GB and 16 GB, respectively. Considering the 2 GB system memory occupancy for each device, we randomly set the resource capacity of edge devices to be within the range of [6 GB, 14 GB]. The edge servers are interconnected, and the network bandwidths between them are randomly generated, averaging 200 Mbps[21].

7.2 Benchmark algorithm

To evaluate the performance, we provide a thorough analysis by comparing our algorithms DLSMD and HMD with the following benchmark schemes:

- **Random:** This algorithm deploys microservice replicas onto edge servers randomly under the constraint of capacity.
- **Greedy:** The algorithm is first arranged in descending order according to the sum of the edge weights of the dag. It greedily deploys them sequentially to edge server nodes that generate the least communication overhead.
- **Gomory-Hu tree:** This algorithm refers to a heuristic algorithm based on the Gomory-Hu tree, an approximation algorithm to solve the min-k-cut problem[48]. This algorithm does not consider the capacity constraint of the edge server node. Therefore, this method will judge whether the partition strategy satisfies the capacity constraint. It will continue partitioning a DAG until it meets the capacity constraint.
- **LSPM:** This method also is a local search-based placement algorithm for min-k-cut which is proposed in Ref. [40]. The strategy used by this algorithm cannot guarantee the direction of convergence in the iterative process. To prevent an endless loop, we limit the algorithm to 10 000 iterations.
- **No-Greedy:** This method assigns numbers to microservices in any order and schedules them to edge servers in ascending order of the number of nodes to minimize the total completion time.
- **P&P[21]:** This method first computes the priority for each microservice considering its computing time and communication time. Nodes with shallower depths have higher priority. Then, it greedily schedules the microservices to edge servers in decreasing order of their priorities to minimize the total completion time.

7.3 Result analysis

Test case 1 (Effect of the number of microservices): In this experiment, we explore the effect of the number of microservices on performance by varying the number of micro-services to {10, 20, 30, 40, 50}. To improve the reliability of the results, we report the average over 100 independent runs. Figure 4 indicates the effect of the number of microservices on the Sum of Weights of Non-cut (SWN). As shown in Fig. 4, SWN increases as the number of microservices increases. Random always presents the worst performance among all benchmarks, which indicates that Kubernetes' default microservice deployment method can not work well in the edge computing system for DAG structure application microservice deployment. DLSMD's performance is the best under all circumstances, regardless of whether the number of microservices is large or small compared with the

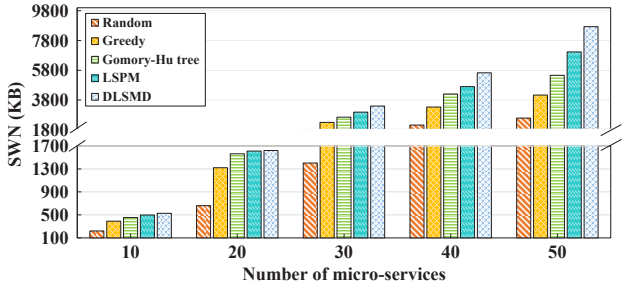
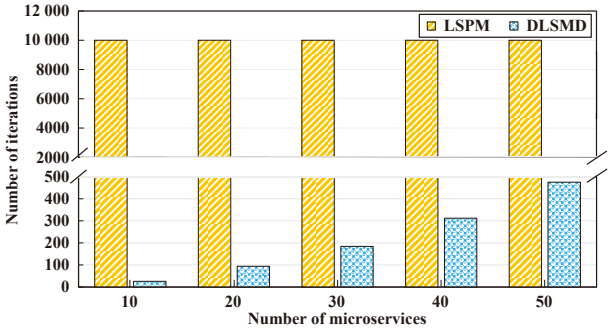
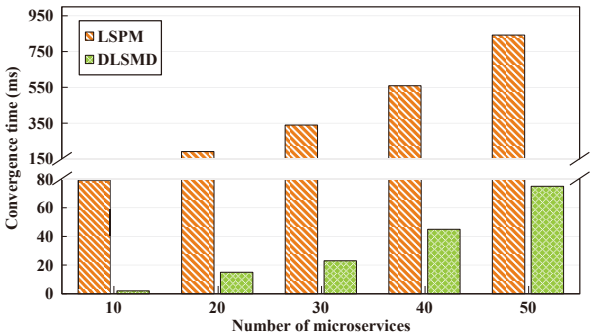


Fig. 4 Effect of the number of microservices on SWN.

Random algorithm, achieving up to 4× speedup. When the number of microservices is relatively small, Greedy can show good performance, but with the increase of the number of microservices, the performance of Greedy gradually decreases. This is because Greedy always greedily chooses the edge server node that generates minimum communication overhead to deploy microservices in each step, which cannot guarantee the overall performance. Gomory-Hu tree and LSPM min-k-cut based methods which can show better performance. Since Gomory-Hu tree and LSPM do not consider the capacity of edge servers or the convergence, they do not perform better than DLSMD. Figure 5 shows the number of iterations and convergence times of local search-based methods LSPM and DLSMD under different numbers of microservices. In Fig. 5a, we can observe that the number of iterations of LSPM is 10 000, which means that the LSPM cannot converge under any circumstance. It is because LSPM does not consider the direction of convergence in algorithm design. The number of iterations of DLSMD increases linearly as the increase of number of microservices. This shows the convergence direction is correct and the algorithm is practical. Figure 5b shows that compared with LSPM, DLSMD's convergence time is insignificant, which linearly increases as the increase of number of microservices. DLSMD's convergence time is less than 80 ms even though the number of microservices is as high as 50, which shows that the algorithm has strong scalability and can be applied to large-scale scenarios.



(a) Comparison of the number of iterations of local search algorithms



(b) Comparison of the convergence time of local search algorithms

Fig. 5 Effects of the number of microservices on the number of iterations and convergence time.

Test case 2 (Effect of the number of DAGs): In Fig. 6, we explore the effect of the number of DAGs on performance. We measure SWN by changing the number of DAGs to {2, 4, 6, 8, 10}. We can observe that with the changing of the number of DAGs, the local search-based algorithms (i.e., LSPM and DLSMD) work better. LSPM performs well when the number of DAGs is small, but declines as the number of DAGs increases. However, DLSMD consistently performs well, regardless of the number of DAGs. This is because DLSMD consolidates all DAGs into a single DAG and provides a convergence guarantee for microservice deployment, which is not considered by LSPM. Next, we investigate the impact of the number of DAGs on the number of iterations and convergence times in Fig. 7. We can observe that the number of iterations of LSPM and the convergence time is much larger than that of DLSMD.

Test case 3 (Effect of the number of microservice requests): In this experiment, we investigate the effect of the number of microservice requests on performance. In Fig. 8, we can find that, as the number of microservice requests increases, the SWN increases. This is because we deploy a replica of the microservice

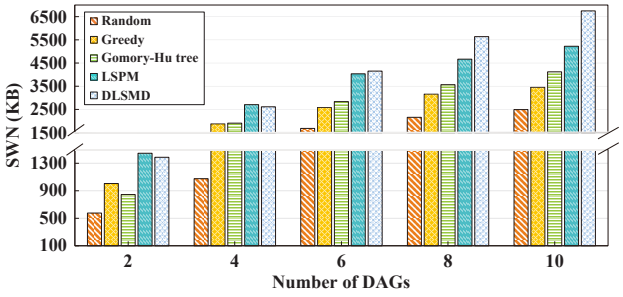
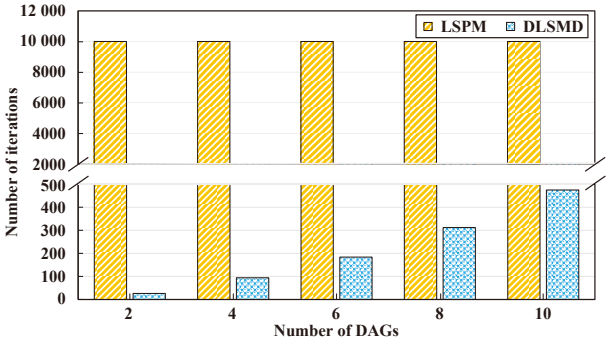
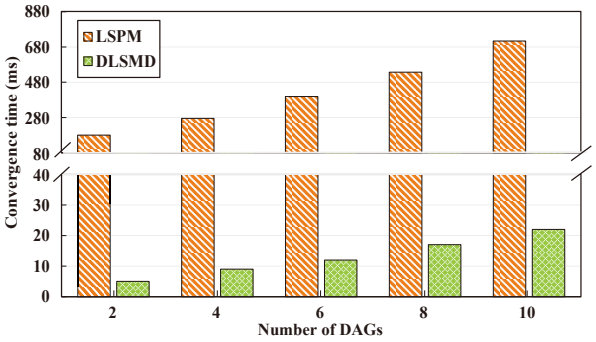


Fig. 6 Effect of the number of DAGs on SWN.



(a) Comparison of the number of iterations of local search algorithms



(b) Comparison of the convergence time of local search algorithms

Fig. 7 Effects of the number of DAGs on the number of iterations and convergence time.

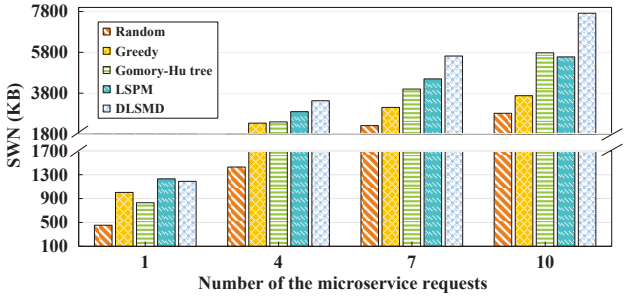
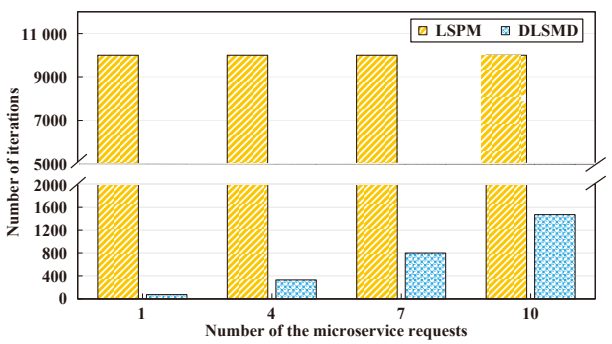
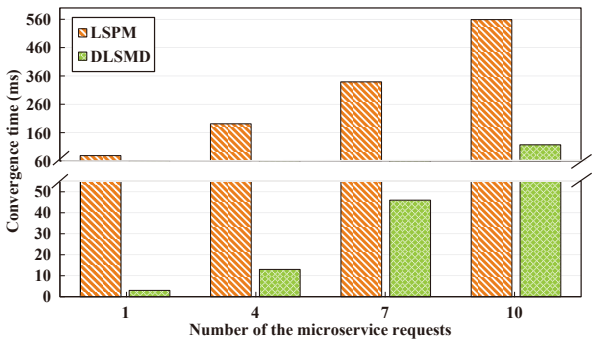


Fig. 8 Effect of the number of microservice requests on SWN.

to each request, which results in increased links between vertices. Therefore more microservice replicas depending on each other may be deployed on the same edge server. At the same time, the increase in requests also leads to an increase in problem complexity. Thus, when requests are {6, 8, 10}, the SWN of the comparison algorithm increases slowly. However, the SWN of DLSMD can increase linearly in high loads, which shows the advantage of the proposed algorithms in this paper. Since the increase in requests will greatly increase the complexity of the problem, the number of iterations of DLSMD and LSPM also increase significantly, as shown in Fig. 9. However, the number of iterations and convergence time of DLSMD are



(a) Comparison of the number of iterations of local search algorithms



(b) Comparison of the convergence time of local search algorithms

Fig. 9 Effects of the number of microservice requests on the number of iterations and convergence time.

much lower than that of LSPM. The reason is that LSPM does not consider the direction of convergence.

Test case 4 (Effect of the number of edge servers): In Fig. 10, we explore the change in SWN by varying the number of edge servers to {5, 10, 15, 20}. We can find that the performance does not increase with the number of edge servers. On the contrary, as the number of edge servers increases, the algorithm performance will reach a peak and then decline. This is because each algorithm distributes the microservice to all servers. When the number of servers reaches a certain level, blindly increasing the number of servers will lead to more dispersed microservice deployment, thus increasing the communication overhead generated by deployment and reducing the non-cut set. But our

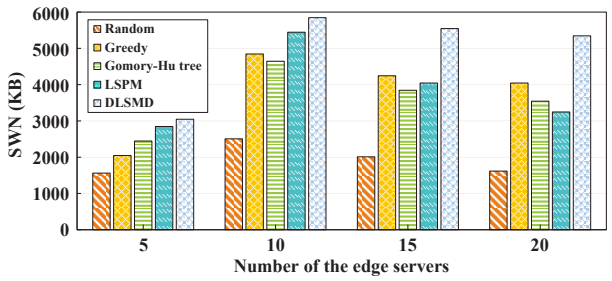


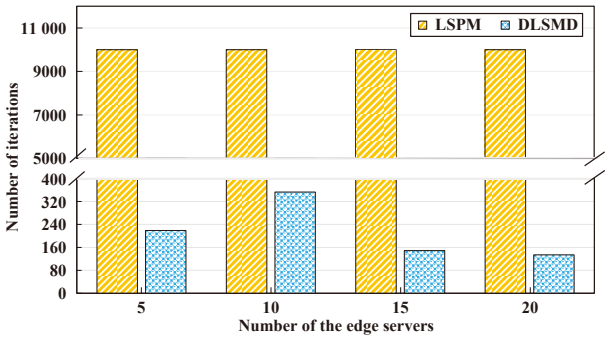
Fig. 10 Effect of the number of edge servers on SWN.

algorithm can always maintain the best performance as the number of edge servers increases.

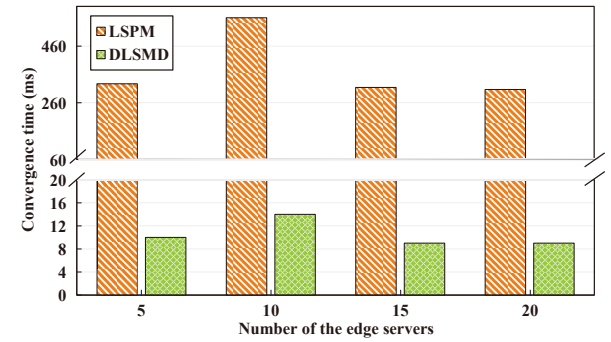
Figure 11 shows that in terms of algorithm complexity, the proposed algorithm DLSMD is still better than LSPM, and the delay is maintained at about 10 ms because the complexity of the algorithm DLSMD is determined by the complexity of DAG, not the number of edge servers.

Test case 5 (Effect of the capacity of edge servers): In Fig. 12, we explore the effect of the capacity of edge servers on SWN. We set up 5 experimental environments and randomly distribute the number of edge servers in {(20–30), (25–35), (30–40), (35–45), (40–50)}. We can find that the performances of all algorithms increase with the increase of capacities of edge server nodes. According to the result, when the server resources are large enough, each algorithm can reach the optimal solution, because all the microservices can be deployed on one server. Greedy performs poorly when there are lower capacities of edge server nodes, but better when there are higher capacities of edge server nodes. This is because when the server capacity is large, Greedy can allocate more replicas of microservice to the same node simultaneously, resulting in greater algorithm improvement. Compared with other algorithms, the performance of DLSMD is always the best, which shows the advantage of our algorithms. As shown in Fig. 13, the number of iterations and convergence time of DLSMD donot change much. This is because the algorithm's complexity is mainly determined by the number of loads in the DAGs and the number of dependencies between vertices.

Test case 6 (Performance of algorithm HMD): In this case, we evaluate the performance of the proposed algorithm HMD, which considers both the computing and communication time in DAG workflows. We first evaluate the effect of the number of microservices on the performance of algorithms Random, No-Greedy, P&P, and HMD in terms of the total completion time, by varying the number of microservices from 30 to 55



(a) Comparison of the number of iterations of local search algorithms



(b) Comparison of the convergence time of local search algorithms

Fig. 11 Effects of the number of edge servers on the number of iterations and convergence time.

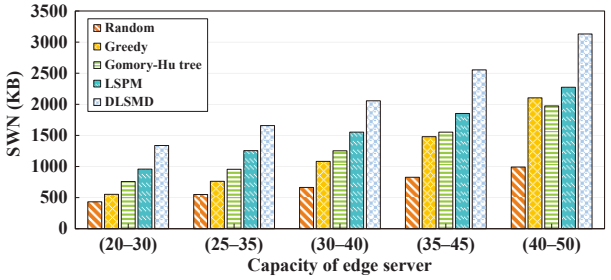
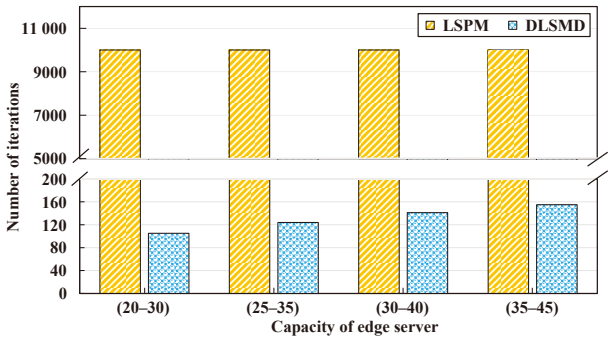
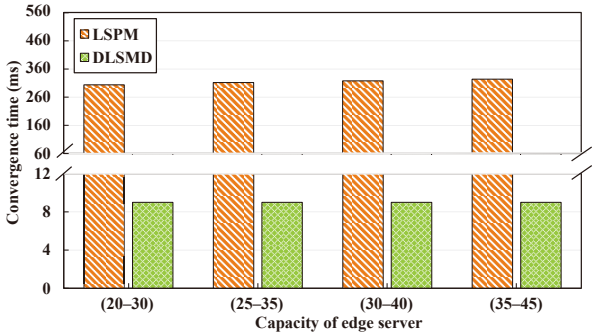


Fig. 12 Effect of the capacity of edge servers on SWN.



(a) Comparison of the number of iterations of local search algorithms



(b) Comparison of the convergence time of local search algorithms

Fig. 13 Effects of the capacity of edge servers on the number of iterations and convergence time.

while fixing the number of edge servers at 5. As shown in Fig. 14a, HMD achieves the lowest completion time compared to Random, No-Greedy, and P&P, with a speedup of up to 2.66×, 1.50×, and 1.49× in total completion time reduction, respectively. We then evaluate the effect of the number of edge servers on the performance of algorithms Random, No-Greedy, P&P, and HMD in terms of the total completion time, by varying the number of edge servers from 5 to 10 while fixing the number of microservices at 55. As shown in Fig. 14b, compared with algorithms Random, No-Greedy, and P&P, the proposed algorithm HMD has the lowest total completion time, achieving a speedup of up to 3.12×, 1.73×, and 1.55×, respectively, in total completion time reduction. The reason is that Random deploys microservices onto edge servers randomly under the resources capacity constraint, thus suffering from worst performance; P&P outperforms No-Greedy because P&P prioritizes placing microservices with shallower depths, allowing more efficient use of server resources. Based on the basic greedy algorithm, HMD can relocate microservices, enabling each one to find a more suitable server. This, in turn, can further reduce the application's completion time and achieve optimal performance.

8 Conclusion

This paper proposes an adaptive scheme to guide microservice deployment for data partition-based applications in the MEC. We model the multi-replica microservice deployment problem as an integer programming problem and prove its NP-hardness. To tackle it, we integrate all container replicas of every application into a single DAG for collaborative deployment. We equivalently convert the optimization objective from the minimization of inter-server traffic to the maximization of intra-server traffic. We propose DLSMD to maximize the intra-server traffic and map each container replica into edge servers. Finally, we theoretically analyze the convergence, time complexity, and approximation ratio to show the performance of DLSMD. We also devise a heuristic algorithm HMD for the more general problem to minimize the total completion time which considers both computing and communication time. Extensive simulation results show that DLSMD and HMD outperform other benchmarks, compared with the Random algorithm, achieving up to 4.00× and 3.12× speedups in terms of the inter-server traffic and total completion time reduction.

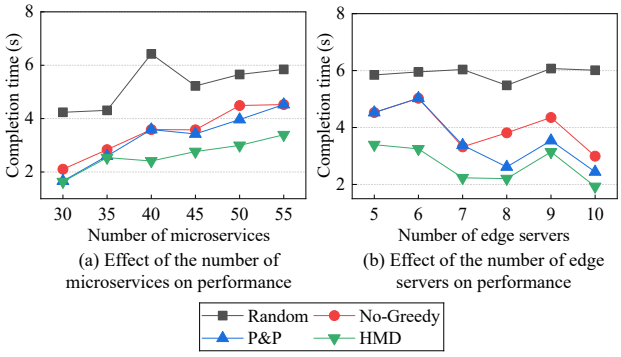


Fig. 14 Effect of the number of microservices and the number of edge servers on performance in terms of the total completion time.

Acknowledgment

This work was supported by the Jiangsu Provincial Frontier Technology Research and Development Program (No. BF2024070). We also thank the Big Data Computing Center of Southeast University in China for providing the experiment environment and computing facility, as well as SF Technology Co., Ltd. for the valuable suggestions during the revision of this manuscript.

References

[1] Z. Xu, D. Liu, W. Liang, W. Xu, H. Dai, Q. Xia, and P. Zhou, Online learning algorithms for offloading augmented reality requests with uncertain demands in MECs, in *Proc. 41st IEEE Int. Conf. Distributed Computing Systems*, Washington, DC, USA, 2021, pp. 1064–1074.

[2] Y. Chen, H. Inaltekin, and M. Gorlatova, Demo abstract: Pixel similarity-based content reuse in edge-assisted virtual reality, in *Proc. IEEE INFOCOM 2022 - IEEE Conf. Computer Communications Workshops*, New York, NY, USA, 2022, pp. 1–2.

[3] M. I. C. Wang, C. H. P. Wen, and H. J. Chao, Hierarchical cooperation and load balancing for scalable autonomous vehicle routing in multi-access edge computing environment, *IEEE Trans. Veh. Technol.*, vol. 72, no. 6, pp. 6959–6971, 2023.

[4] Q. Chen, Z. Zheng, C. Hu, D. Wang, and F. Liu, On-edge multi-task transfer learning: Model and practice with data-driven task allocation, *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1357–1371, 2020.

[5] S. Chen, L. Wang, and F. Liu, Optimal admission control mechanism design for time-sensitive services in edge computing, in *Proc. IEEE INFOCOM 2022 - IEEE Conf. Computer Communications*, London, UK, 2022, pp. 1169–1178.

[6] B. Gao, Z. Zhou, F. Liu, F. Xu, and B. Li, An online framework for joint network selection and service

placement in mobile edge computing, *IEEE Trans. Mobile Comput.*, vol. 21, no. 11, pp. 3836–3851, 2022.

[7] J. Wu, L. Wang, Q. Pei, X. Cui, F. Liu, and T. Yang, HiTDL: High-throughput deep learning inference at the hybrid mobile edge, *IEEE Trans. Parall. Distrib. Syst.*, vol. 33, no. 12, pp. 4499–4514, 2022.

[8] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, Edge intelligence: Paving the last mile of artificial intelligence with edge computing, *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[9] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, Resource scheduling in edge computing: A survey, *IEEE Commun. Surv. Tutor.*, vol. 23, no. 4, pp. 2131–2165, 2021.

[10] W. Zhang, Z. He, L. Liu, Z. Jia, Y. Liu, M. Gruteser, D. Raychaudhuri, and Y. Zhang, Elf: Accelerate high-resolution mobile deep vision with content-aware parallel offloading, in *Proc. 27th Annu. Int. Conf. Mobile Computing and Networking*, New Orleans, LA, USA, 2021, pp. 201–214.

[11] X. Wang, Z. Yang, J. Wu, Y. Zhao, and Z. Zhou, EdgeDuet: Tiling small object detection for edge assisted autonomous mobile vision, in *Proc. IEEE INFOCOM 2021 - IEEE Conf. Computer Communications*, Vancouver, Canada, 2021, pp. 1–10.

[12] S. Jiang, Z. Lin, Y. Li, Y. Shu, and Y. Liu, Flexible high-resolution object detection on edge devices with tunable latency, in *Proc. 27th Annu. Int. Conf. Mobile Computing and Networking*, New Orleans, LA, USA, 2021, pp. 559–572.

[13] L. Liu, H. Li, and M. Gruteser, Edge assisted real-time object detection for mobile augmented reality, in *Proc. 25th Annu. Int. Conf. Mobile Computing and Networking*, Los Cabos, Mexico, 2019, p. 25.

[14] L. Gu, Z. Chen, H. Xu, D. Zeng, B. Li, and H. Jin, Layer-aware collaborative microservice deployment toward maximal edge throughput, in *Proc. IEEE INFOCOM 2022 - IEEE Conf. Computer Communications*, London, UK, 2022, pp. 71–79.

[15] J. Shi, H. Zhang, Z. Tong, Q. Chen, K. Fu, and M. Guo, Nodens: Enabling resource efficient and fast QoS recovery of dynamic microservice applications in datacenters, in *Proc. 2023 USENIX Annu. Technical Conf.*, Boston, MA, USA, 2023, pp. 403–417.

[16] S. Luo, H. Xu, C. Lu, K. Ye, G. Xu, L. Zhang, Y. Ding, J. He, and C. Xu, Characterizing microservice dependency and performance: Alibaba trace analysis. in *Proc. ACM Symp. Cloud Computing*, Seattle, WA, USA, 2021, pp. 412–426.

[17] V. M. Bhasi, J. R. Gunasekaran, P. Thinakaran, C. S. Mishra, M. T. Kandemir, and C. Das, Kraken: Adaptive container provisioning for deploying dynamic DAGs in serverless platforms, in *Proc. ACM Symp. Cloud Computing*, Seattle, WA, USA, 2021, pp. 153–167.

[18] K. Fu, W. Zhang, Q. Chen, D. Zeng, and M. Guo, Adaptive resource efficient microservice deployment in cloud-edge continuum, *IEEE Trans. Parall. Distrib. Syst.*, vol. 33, no. 8, pp. 1825–1840, 2022.

[19] S. Wang, Z. Ding, and C. Jiang, Elastic scheduling for microservice applications in clouds, *IEEE Trans. Parall. Distrib. Syst.*, vol. 32, no. 1, pp. 98–115, 2021.

[20] X. Zhu, Y. Li, L. Yao, Z. Qi, Y. Xu, P. Wang, W. Wang, and X. Zhu, On optimizing traffic scheduling for multi-replica containerized microservices, in *Proc. 52nd Int. Conf. Parallel Processing*, Salt Lake City, UT, USA, 2023, pp. 358–368.

[21] Y. Li, L. Gu, Z. Qu, L. Tian, and D. Zeng, On efficient zygote container planning and task scheduling for edge native application acceleration, in *Proc. IEEE INFOCOM 2024 - IEEE Conf. Computer Communications*, Vancouver, Canada, 2024, pp. 2259–2268.

[22] C. Hu, W. Bao, D. Wang, and F. Liu, Dynamic adaptive DNN surgery for inference acceleration on the edge, in *Proc. IEEE INFOCOM 2019 - IEEE Conf. Computer Communications*, Paris, France, 2019, pp. 1423–1431.

[23] A. Moubayed, A. Shami, P. Heidari, A. Larabi, and R. Brunner, Edge-enabled V2X service placement for intelligent transportation systems, *IEEE Trans. Mobile Comput.*, vol. 20, no. 4, pp. 1380–1392, 2021.

[24] Y. Li and S. Wang, An energy-aware edge server placement algorithm in mobile edge computing, in *Proc. 2018 IEEE Int. Conf. Edge Computing*, San Francisco, CA, USA, 2018, pp. 66–73.

[25] T. Ouyang, Z. Zhou, and X. Chen, Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing, *IEEE J. Select. Areas Commun.*, vol. 36, no. 10, pp. 2333–2345, 2018.

[26] S. Yang, F. Li, S. Trajanovski, X. Chen, Y. Wang, and X. Fu, Delay-aware virtual network function placement and routing in edge clouds, *IEEE. Trans. Mobile Comput.*, vol. 20, no. 2, pp. 445–459, 2021.

[27] Y. Niu, F. Liu, and Z. Li, Load balancing across microservices, in *Proc. IEEE INFOCOM 2018 - IEEE Conf. Computer Communications*, Honolulu, HI, USA, 2018, pp. 198–206.

[28] W. Bao, D. Yuan, B. B. Zhou, and A. Y. Zomaya, Prune and plant: Efficient placement and parallelism of virtual network functions, *IEEE Trans. Comput.*, vol. 69, no. 6, pp. 800–811, 2020.

[29] P. Lin, Z. Shi, Z. Xiao, C. Chen, and K. Li, Latency-driven model placement for efficient edge intelligence service, *IEEE Trans. Serv. Comput.*, vol. 15, no. 2, pp. 591–601, 2022.

[30] K. Y. Chen, Y. Xu, K. Xi, and H. J. Chao, Intelligent virtual machine placement for cost efficiency in geo-distributed cloud systems, in *Proc. 2013 IEEE Int. Conf. Communications*, Budapest, Hungary, 2013, pp. 3498–3503.

[31] K. Huang and B. Shen, Service deployment strategies for efficient execution of composite SaaS applications on cloud platform, *J. Syst. Softw.*, vol. 107, pp. 127–141, 2015.

[32] Y. Wang, P. Lu, W. Lu, and Z. Zhu, Cost-efficient virtual network function graph (vNFG) provisioning in multidomain elastic optical networks, *J. Lightwave Technol.*, vol. 35, no. 13, pp. 2712–2723, 2017.

[33] Y. Han, S. Shen, X. Wang, S. Wang, and V. C. M. Leung, Tailored learning-based scheduling for Kubernetes-oriented edge-cloud system, in *Proc. IEEE INFOCOM*

2021 - IEEE Conf. Computer Communications, Vancouver, Canada, 2021, pp. 1–10.
[34] J. Park, U. Choi, S. Kum, J. Moon, and K. Lee, Accelerator-aware Kubernetes scheduler for DNN tasks on edge computing environment, in *Proc. 2021 IEEE/ACM Symp. Edge Computing*, San Jose, CA, USA, 2021, pp. 438–440.
[35] F. Rossi, V. Cardellini, F. Lo Presti, and M. Nardelli, Geo-distributed efficient deployment of containers with Kubernetes, *Comput. Commun.*, vol. 159, pp. 161–174, 2020.
[36] G. Qu, Z. Lin, F. Liu, X. Chen, and K. Huang, TrimCaching: Parameter-sharing AI model caching in wireless edge networks, in *Proc. 44th Int. Conf. Distributed Computing Systems*, Jersey City, NJ, USA, 2024, pp. 36–46.
[37] L. Pan, L. Wang, S. Chen, and F. Liu, Retention-aware container caching for serverless edge computing, in *Proc. IEEE INFOCOM 2022 - IEEE Conf. Computer Communications*, London, UK, 2022, pp. 1069–1078.
[38] P. Jin, X. Fei, Q. Zhang, F. Liu, and B. Li, Latency-aware VNF chain deployment with efficient resource reuse at network edge, in *Proc. IEEE INFOCOM 2020 - IEEE Conf. Computer Communications*, Toronto, Canada, 2020, pp. 267–276.
[39] W. Zhu and C. Guo, Local search approximation algorithms for the complement of the min-k-cut problems, Research Report, Center for Discrete Mathematics and Theoretical Computer Science,Fuzhou University, Fuzhou, China, https://optimization-online.org/2010/07/2673/, 2025.
[40] D. R. Gaur, R. Krishnamurti, and R. Kohli, The capacitated max *k*-cut problem, *Math. Program.*, vol. 115, no. 1, pp. 65–72, 2008.
[41] M. Pirlot, General local search methods, *Eur. J. Oper. Res.*, vol. 92, no. 3, pp. 493–511, 1996.
[42] T. Shi, H. Ma, G. Chen, and S. Hartmann, Location-aware and budget-constrained service deployment for composite applications in multi-cloud environment, *IEEE Trans. Parall. Distrib. Syst.*, vol. 31, no. 8, pp. 1954–1969, 2020.
[43] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, Fast adaptive task offloading in edge computing based on meta reinforcement learning, *IEEE Trans. Parall. Distrib. Syst.*, vol. 32, no. 1, pp. 242–253, 2021.
[44] J. Yan, S. Bi, and Y. J. A. Zhang, Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach, *IEEE Trans. Wireless Commun.*, vol. 19, no. 8, pp. 5404–5419, 2020.
[45] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, Offloading tasks with dependency and service caching in mobile edge computing, *IEEE Trans. Parall. Distrib. Syst.*, vol. 32, no. 11, pp. 2777–2792, 2021.
[46] L. Liu, H. Tan, S. H. C. Jiang, Z. Han, X. Y. Li, and H. Huang, Dependent task placement and scheduling with function configuration in edge computing, in *Proc. 27th Int. Symp. Quality of Service*, Phoenix, AZ, USA, 2019, pp. 1–10.
[47] X. Shang, Y. Mao, Y. Liu, Y. Huang, Z. Liu, and Y. Yang, Online container scheduling for data-intensive applications in serverless edge computing, in *Proc. IEEE INFOCOM 2023 - IEEE Conf. Computer Communications*, New York, NY, USA, 2023, pp. 1–10.
[48] X. Zhang, Y. Zhao, S. Guo, and Y. Li, Performance-aware energy-efficient virtual machine placement in cloud data center, in *Proc. 2017 IEEE Int. Conf. Communications*, Paris, France, 2017, pp. 1–7.

Zhaowu Huang received the BEng degree from Nanjing University of Science and Technology, China, in 2018. He is currently a PhD candidate at Southeast University, China. His research interests include edge computing and edge intelligence.

Fang Dong is currently a professor at School of Computer Science and Engineering, Southeast University, China. He received the BEng and MEng degrees in computer science from Nanjing University of Science & Technology, China, in 2004 and 2006, respectively, and the PhD degree in computer science from Southeast University, China, in 2011. His current research interests include edge intelligence, cloud computing and industrial Internet. He is a member of IEEE and ACM. He served as the co-chair of ACM Nanjing Chapter and the general secretary of ACM SIGCOMM China.

Wenlong Ruan received the BEng degree from Changsha University of Science & Technology, China, in 2019. He is currently a master student at Southeast University, China. His main research interest is edge computing.

Xiaolin Guo received the BEng degree from Nanjing University of Science and Technology, China, in 2018. She is currently a PhD candidate at Southeast University, China. Her current research interests are edge computing, in-network computing, and edge intelligence.

Jinghui Zhang received the BEng degree from Southeast University, China, in 2005, and the PhD degree in computer science from Southeast University, China, in 2014, where he is currently an associate professor at School of Computer Science and Engineering. His research interests include edge computing, distributed machine learning, and cloud computing.