Efficient Microservice Deployment with Dependencies in Multi-Access Edge Computing

Shuaibing Lu
Faculty of Information Technology
Beijing University of Technology
Beijing, China
lushuaibing@bjut.edu.cn

Ran Yan
Faculty of Information Technology
Beijing University of Technology
Beijing, China
yanran@emails.bjut.edu.cn

Jie Wu
Center for Networked Computing
Temple University
Philadelphia, USA
jiewu@temple.edu

***Abstract*—In the context of mobile edge computing, efficiently deploying microservices to reduce finish time and enhance user service quality is a challenging task. However, existing research still has certain deficiencies in considering microservice deployment within edge server clusters and communication link constraints. To address this issue, we propose three microservice deployment strategies by offering flexibility and adaptability for various application scenarios. We initially consider two straightforward scenarios: one with unlimited storage resources under the bandwidth constraint, and the other with unlimited bandwidth resources under the storage constraint. For each of these two scenarios, we introduce a novel enhanced graph construction method and design two optimal solutions. For Scenario 3, which involves complex constraints on server capacity, computational capability, and communication resources, we present an optimization method based on main path partitioning and the simulated annealing algorithm. We effectively tackle challenges arising from server capacity, computational capability, and communication resource limitations. Across multiple experimental setups, our approach significantly improves microservice deployment efficiency and overall performance compared to traditional strategies.**

***Index Terms*—microservice deployment, dependency, high-efficient, multi-access edge computing.**

I. Introduction

With the widespread adoption of mobile devices and the continuous emergence of mobile applications, traditional cloud computing faces a range of challenges, such as high latency, network congestion, and extensive data transmission [20]. Multi-access edge computing, as a flexible and scalable computing platform, pushes computation and data processing to the network edge, closer to users and devices [16]. This effectively reduces the distances between data transmissions on the network, which significantly reduces latency and improves responsiveness. The limitations of traditional monolithic applications in terms of scalability and flexibility have led to the emergence of microservice architecture, a lightweight and highly flexible architectural pattern [12]. By decomposing complex monolithic applications into small, autonomous service units, modularity, scalability, and maintainability are improved. This architecture is widely employed in constructing distributed systems and cloud-native applications. However, it is worth noting that effectively deploying microservices

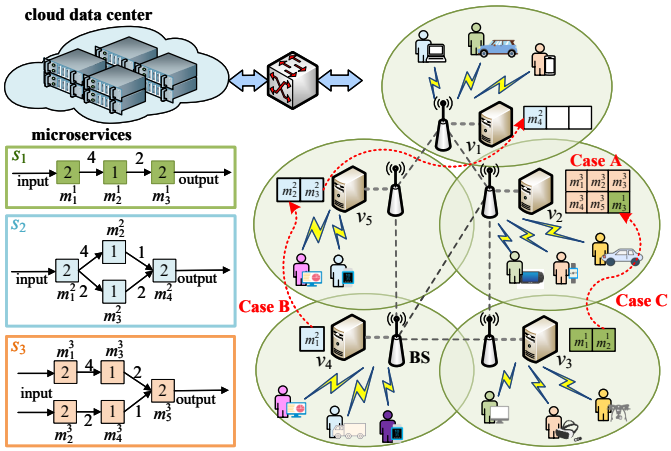


Fig. 1. An illustrating example.

in large distributed environments, both in the cloud and on edge devices, still presents a range of challenges. This is especially true when it comes to deploying microservices with dependencies, where the complexity and diversity of the problem exacerbate the challenge. There are two main issues: (i) How can complex dependencies among microservices be effectively dealt with to improve the overall efficiency? (ii) How to balance the trade-off between processing and transmission time for optimal deployment without overwhelming the resource constraints? Therefore, efficiently managing complicated inter-dependencies between microservices, optimizing deployment strategies, and ensuring overall system performance and stability become crucial research topics.

In this paper, we delve into the optimization of deployment strategies for microservices with complex dependencies in multi-access edge computing, thoroughly examining the problem of dependent microservice deployment across different scenarios. We investigate the effective deployment of dependent microservices and aim to reduce the makespan while enhancing the overall system performance and resource utilization that provide more advantages and opportunities for microservices within the context of mobile edge computing. Our major contributions are summarized as follows:

- We investigate the microservice deployment problem

with dependencies by formulating to minimize the makespan of multiple services under the resources (storage, computing, and communication) constraints in multi-access edge computing.
- We propose three strategies for deploying microservices that provide flexibility and adaptability for different application scenarios. We first consider two straightforward scenarios: one with unlimited storage resources under the bandwidth constraint, and the other with unlimited bandwidth resources under the storage constraint. For each of these two scenarios, we introduce a novel enhanced graph construction method and design two optimal solutions.
- We then consider a more complicated scenario with resource constraints on storage, computing, and communication. We devise a feasible solution by introducing an effective embedding method based on the novel definitions of the main path and the preferred server extracted based on the topology features of the services and the edge environment, respectively.
- We conducted extensive experiments to compare our strategies with several baselines based on the China Telecom Shanghai Company dataset, which was constructed by the geographic information of 3,233 base stations. The results are shown from different perspectives to provide conclusions. Extensive experiments demonstrate that our methods yield favorable results while reducing time complexity under different scenarios.

The remainder of this paper is organized as follows. Section II surveys related work. Section III introduces the model and presents the problem. Section IV explores three distinct dependent microservice deployment strategies for different scenarios. Section V presents the experimental results. Finally, Section VI provides a summary of the entire paper.

II. Related work

Microservices architecture, known for its lightweight and flexible nature, finds wide applications in building distributed systems and cloud-native applications. To address these challenges, researchers have introduced various innovative methods and strategies. Ding et al. [1] proposed a solution to the microservice placement problem using an improved genetic algorithm to obtain the same throughput at a lower cost. Tang et al. [2] proposed an adaptive dynamic deployment optimization method to minimize resource consumption costs. Carrusca et al. [3] proposed a solution to automatically deploy microservices in the edge to reduce latency. Wang et al. [4] proposed an offline algorithm to achieve optimal microservices coordination when future system information is available, reducing overall latency. However, these methods frequently fail to take into account the latency problem brought on by dependencies between microservices.

Since the popularity of microservices architecture, some works focuses on microservice deployment with dependencies. Menouer et al. [5] preserve internal dependencies between microservices by defining an ordered sequence. Zhao et al. [6] determined deployment configurations for each microservice by considering uncertainty in requests. Niu et al. [18] proposed a message queue-based chain-oriented load balancing algorithm to minimizing response time. Guerrero et al. [7] proposed a common genetic algorithm for multi-objective problems to reduce service costs, repair time, and delay overhead. Deng et al. [8], [9] introduces a dependent function embedding algorithm to reduce makespan by determining the optimal split for each data stream.Pallewatta et al. [10] proposed a particle swarm optimization based on multi-objective sets to optimize completion time. Liao et al. [11] prioritized tasks and handled application allocation and scheduling problems jointly in an online manner based on priority estimation. Lv et al. [19] proposed a graph convolution network deployment framework based on reinforcement learning to ensure user QoS. These efforts achieved promising results, however, they neglected the impact of server heterogeneity on the resources in edge environments. In this paper, we comprehensively investigate the problem of dependent microservice deployment across various scenarios and focus on the optimization of methods for deploying microservices with complicated dependencies in multi-access edge computing.

III. Problem Formulation

A. System model

We consider a three-layer network architecture, as represented in Figure 1, which consists of the cloud data center, edge servers, and end users. Given a substrate topology of edge network which is modeled as a weighted undirected graph $\mathbf{G}(V, L)$, where $V = \{v_k\}$ and $L = \{l_{(v_k,v_q)}\}$ represent the sets of edge servers and links, respectively. Here, we use v_k to denote the k-th edge server, and $l_{(v_k,v_q)}$ represents the communication link between servers v_k and v_q. The computing capability of edge server v_k is represented as $c_{(v_k)}$, measured in gflop/s, and the communication capacity of $l_{(v_k,v_q)}$ is denoted by $b_{(v_{k,q})}$, measured in GB/s.Each edge server has a storage capacity, denoted by $\phi_{(v_k)}$, representing the maximum number of microservices it can accommodate. In addition, we assume that the services required by the users have been originally provisioned in the cloud data center [13], which is represented by set $\mathbf{S} = \{S_h\}$. Here, we use S_h to denote the h-th service that consists a set of microservices $M_h = \{m_i^h\}$ and directed links $E_h = \{e^h_{m_i \to m_j}\}$, i.e., $S_h = \{M_h, E_h\}$. Let m_i^h represent the i-th microservice of S_h, and the required processing capability of m_i^h is denoted as $q_{m_i^h}$, measured in $gflops$. Let $e^h_{m_i \to m_j}$ represent the dependency between m_i^h and m_j^h, and we use $r^h_{m_i \to m_j}$ to denote the corresponding data flow size between microservices m_i^h and m_j^h, representing the weight of directed edge $e^h_{m_i \to m_j}$, measured in GB.

B. Computation and Communication Models

We use $d_c(m_i^h)$ to represent the processing time of microservice m_i^h on an edge server. The processing time reflects

the time needed for microservice execution on the edge server. It is calculated by dividing the computation workload $q_{m_i^h}$ of microservice when deployed on an edge server by the computing capability $c_{(v_k)}$ of that server. We use $x(i,k)$ to indicate whether microservice m_i^h is deployed on edge server v_k. If m_i^h is deployed on v_k, then $x(i,k) = 1$; otherwise, $x(i,k) = 0$. The computation time is given by:

$$d_c(m_i^h) = x(i,k) \cdot q_{m_i^h}/c_{(v_k)} \tag{1}$$

Here, we use $d_l(e^h_{m_i \to m_j})$ to represent the transferring time of two dependent microservices, where m_i^h is the predecessor of m_j^h. The transferring time is given by:

$$d_l(e^h_{m_i \to m_j}) = y(i,j) \cdot r^h_{m_i \to m_j}/b_{(v_k,v_q)} \tag{2}$$

We use $y(i,j)$ to indicate whether microservice m_i^h and m_j^h are deployed on the same edge server. When two dependent microservices are deployed on different edge servers, data needs to be transmitted through links of communication, and $y(i,j) = 1$. On the contrary, if two dependent microservices are deployed on the same edge server, $y(i,j) = 0$, which means that server-to-server data transfers are seamless and result in $d_l(e^h_{m_i \to m_j}) = 0$.

C. Problem Formulation

In this paper, we focus on finding an efficient strategy that minimizes the makespan of services in set **S** under the constraints, which is determined by the part with the longest completion time. We use $f(m_j^h)$ to denote the completion time of microservice m_j^h which has a predecessor m_i^h, i.e., $m_i^h \to m_j^h$. It depends on the completion time $f(m_i^h)$ of the predecessor m_i^h, the computation time $d_c(m_j^h)$ of m_j^h, and the transmission time $d_l(e^h_{m_i \to m_j})$ for data transferred from predecessor m_i^h to m_j^h. However, it is worth noting that there may be multiple predecessor microservices in a service with complex dependencies. Thus, the value of $f(m_j^h)$ is determined by the path of precedence with the maximum completion time can be calculated as:

$$f(m_j^h) = \max_{\forall i,j}\{f(m_i^h) + d_c(m_j^h) + d_l(e^h_{m_i \to m_j})\}. \tag{3}$$

Here, we define T_h as the makespan of service S_h, which depends on the maximum completion time of all microservices in S_h, where

$$T_h = \max_{\forall j, m_j^h \in S_h}\{f(m_j^h)\}, \tag{4}$$

and the makespan of services in set **S** is given by

$$\mathbf{T} = \max_{S_h \in S}\{T_h\}. \tag{5}$$

Therefore, the problem formulation is shown as follows:

$$\mathbf{P1}: \text{minimize}_{\forall i,j} \quad \mathbf{T} \tag{6}$$

$$\text{s.t.} \sum_{k=1}^{n} x(i,k) = 1, \quad \forall i \tag{7}$$

$$\sum_{i=1}^{n} x(i,k) \le \phi_{(v_k)}, \quad \forall k \tag{8}$$

$$b_{(v_k,v_q)} \ll \tau \tag{9}$$

$$x(i,k) \in \{0,1\}, \quad y(i,j) \in \{0,1\}, \quad \forall i, \forall j, \forall k. \tag{10}$$

P1 is the objective function that centers upon minimizing the makespan of services, and equations (6) to (10) are the constraints. Equation (7) signifies that each microservice can only be allocated to a single edge server. Equation (8) states that the number of microservices processed on an edge server cannot exceed its storage capacity. Equation (9) represents the constraint imposed by the communication bandwidth, where τ is the threshold determined by the microservices and servers. Equation (10) specifies the decision of microservice m_i^h that whether deployed on edge server v_k, where $x(i,k) \in \{0,1\}$, and the status that whether m_i^h and m_j^h co-located on edge server v_k, where $y(i,j) \in \{0,1\}$.

***Definition** 1 (Optimal Microservice Deployment with Dependencies (OMDD) problem):* Given the distribution of microservice **S**, the topology of edge network **G**, an OMDD problem is how to find a strategy for microservice in **S** to minimize **P1** under the constraints (7)-(10).

IV. Algorithm Design

A. enhanced graph construction

In this subsection, we consider to deal with the problem of prioritization among multiple services. We introduce a novel enhanced graph construction method to obtain the makespan in equation (5) where multiple services in **S** are deployed in parallel. We define the enhanced graph as $\hat{\mathbb{I}}$, and the transformation involves creating one virtual source m_s and one virtual destination m_d to connect all services. We suppose that the required processing capacities of m_s and m_d are all 0, where $q_{m_s} = 0$ and $q_{m_d} = 0$. To be more precise, we construct $H_h = \{m^h_\omega|_{m^h_\omega \in M_h}\}$ as the set of starting nodes in service S_h, and $\mathbf{H} = \{H_h\}$ represents the set of starting nodes of all services. Then we add directed edges $e_{m_x \to m^h_\omega}$ connecting between virtual source m_s and all starting nodes in **H** of all services $\forall S_h \in \mathbf{S}$. Then we construct $D_h = \{m^h_\varpi|_{m^h_\varpi \in M_h}\}$ as the set of ending nodes in service S_h, and $\mathbf{D} = \{D_h\}$ represents the set of ending nodes of all services. We add directed edges $e_{m^h_\varpi \to m_d}$ connecting between all ending nodes m_ϖ in **D** and the virtual destination m_d. We then give values to the directed edges which represent the required data flow size, where $r_{m_x \to m^h_\omega} = 0$ and $r_{m^h_\varpi \to m_d} = 0$. As the newly added nodes and edges do not change the dependencies and the completion time of microservices for the services, therefore, the makespan of the enhanced graph $\hat{\mathbb{I}}$ has an equivalence on the set of services **S**.

B. Scenario 1: OMDD with no storage constraint

We first consider one scenario on minimizing **P1** with no storage constraint, which is relaxing the value of $\phi_{(v_k)}$ in equation (8), i.e., $\phi_{v_k} \ge \sum_{h=1}^{|S|} |M_h|$ for all k and h. Thus, the initial optimization problem has changed to become how to balance the computing and communication resources, which is formulated as follows.

$$\sum_{i=1}^{n} x(i,k) \le \phi_{(v_k)}, \phi_{v_k} \ge \sum_{h=1}^{|S|} |M_h|, \quad \forall k \tag{11}$$

On the basis of the interaction, we propose a greedy-based method of Algorithm 1 which proved to be optimal. We use the enhanced graph $\hat{\mathbb{I}}$, and edge network $\mathbf{G}$ as the input. The output is the microservices deployment strategy $\mathbf{X}$ and the makespan $\mathbf{T}$. Firstly, for each server in set V, we calculate the sum of ϕ_{v_k} in lines 1 to 2. Then we calculate the total number of microservices in lines 3 to 4. We determine whether the total storage sum_ϕ can accommodate all microservices sum_m in line 5. If the edge network $\mathbf{G}$ can accommodate all microservices, where $sum_\phi \geq sum_m$, we sort V in descending order according to the server's computing capability $c_{(v_k)}$ to find the server with the highest computing capability in line 6. Then, in lines 9 to 12, we start to deploy the microservices. For each m_i in $\hat{\mathbb{I}}$, we deploy the m_i on the server v_0 in line 10. Based on that, update the deployment list $\mathbf{X}$ and calculate the $\mathbf{T}$ in lines 11 to 12. Finally, the microservices deployment strategy $\mathbf{X}$ and the makespan $\mathbf{T}$ are returned in line 13.

***Theorem** 1:* OMDD-US is an optimal solution for solving **P1** under the constraints (7), (9)-(11).

Proof: We prove this theorem by contradiction. We assume that the completion time for placing microservices separately denoted as $\mathbf{T}_s$ is lower than that of merging them as a whole $\mathbf{T}_t$, i.e., $\mathbf{T}_s < \mathbf{T}_t$. Suppose there are two edge servers v_1 and v_2, where $c_{(v_1)} \geq c_{(v_2)}$. We first consider the simplest case of a chain-like microservice graph with only two microservices m_x and m_y. When microservices are deployed separately, m_x is located on edge server v_1, and m_y is located on edge server v_2. Given this, the completion time $\mathbf{T}_s$ for separate deployment is: $\mathbf{T}_s = q_{m_x}/c_{(v_1)} + q_{m_y}/c_{(v_2)} + r_{m_x \to m_y}/b_{v_1,v_2}$. Then, we consider the case that placing all microservices on the same edge server v_1, which has $\mathbf{T}_t = q_{m_x}/c_{(v_1)} + q_{m_y}/c_{(v_1)}$. Since we suppose $c_{(v_1)} \geq c_{(v_2)}$, it follows that $q_{m_y}/c_{(v_2)} \geq q_{m_y}/c_{(v_1)}$. We calculate the difference between $\mathbf{T}_s$ and $\mathbf{T}_t$, where $\mathbf{T}_s - \mathbf{T}_t = (q_{m_y}/c_{(v_2)} - q_{m_y}/c_{(v_1)})) + r_{m_x \to m_y}/b_{v_1,v_2}$. Due to the fact that $r_{m_x \to m_y}/b_{v_1,v_2} \geq 0$, we have $\mathbf{T}_s - \mathbf{T}_t \geq 0$, i.e., $\mathbf{T}_s \geq \mathbf{T}_t$, which contradicts our assumption. Then, we consider a more realistic case of a DAG-based microservice graph with only complex dependencies. We assume that there exists a path $m_x \to m_y \to m_z$ with the maximum required processing capability of services $\mathbf{S}$, and the makespan $\mathbf{T}_t = q_{m_x}/c_{(v_1)} + q_{m_y}/c_{(v_1)} + q_{m_z}/c_{(v_1)}$ for the case that placing all microservices on the same edge server v_1. Assume that if these microservices are deployed separately, where m_x is deployed on the server v_1, and m_y and m_z are deployed on the server v_2. The completion time $\mathbf{T}_s$ will be $\mathbf{T}_s = q_{m_x}/c_{(v_1)} + q_{m_y}/c_{(v_2)} + q_{m_y}/c_{(v_2)} + r_{m_x \to m_y}/b_{v_1,v_2}$. Additionally, we calculate the difference between $\mathbf{T}_s$ and $\mathbf{T}_t$, where $\mathbf{T}_s - \mathbf{T}_t = ((q_{m_y} + q_{m_z})/c_{(v_2)} - (q_{m_y} + q_{m_z})/c_{(v_1)}) + r_{m_x \to m_y}/b_{v_1,v_2}$. Since we suppose $c_{(v_1)} \geq c_{(v_2)}$, it follows that $(q_{m_y} + q_{m_z})/c_{(v_2)} \geq (q_{m_y} + q_{m_z})/c_{(v_1)}$. Furthermore, due to the fact that $r_{m_x \to m_y}/b_{v_1,v_2} \geq 0$, we are able to deduce that $\mathbf{T}_s - \mathbf{T}_t \geq 0$, i.e., $\mathbf{T}_s \geq \mathbf{T}_t$, which contradicts our assumption. Therefore, we can obtain that OMDD-US can minimize **P1** under the constraints (7), (9)-(11). ■

Algorithm 1 OMDD with unlimited-storage (OMDD-US)

Require: $\hat{\mathbb{I}}$, $\mathbf{G}$.
Ensure: $\mathbf{X}$, $\mathbf{T}$.
1: **for** $v_k \leftarrow 1$ to $|V|$ **do**
2: $\quad sum_\phi \leftarrow sum_\phi + \phi_{v_k}$;
3: **for** $m_i \leftarrow 1$ to $\hat{\mathbb{I}}$ **do**
4: $\quad sum_m \leftarrow sum_m + 1$;
5: **if** $sum_\phi \geq sum_m$ **then**
6: $\quad V \leftarrow$ Update with order by $v_k = \arg\max\{c_{(v_k)}\}$;
7: **else**
8: $\quad$ **break**
9: $\quad$ **for** each m_i in $\hat{\mathbb{I}}$ **do**
10: $\quad\quad$ Place microservice m_i on edge server v_0.
11: $\quad\quad$ Update the deployment list $\mathbf{X}$.
12: $\quad\quad \mathbf{T} \leftarrow \mathbf{T} + \ q_{m_i}/c_{(v_1)}$
13: **Return** $\mathbf{X}$, $\mathbf{T}$.

C. Scenario 2: OMDD with no communication constraint

In this subsection, we investigate one scenario on minimizing **P1** with no bandwidth constraint, which removes equation (9), i.e., $b_{(v_k,v_q)} \geq \tau$. Thus, the initial optimization problem has changed to become how to balance the computing and storage resources. On the basis of the interaction, we propose a greedy-based method of Algorithm 2 which proved to be optimal. We use the enhanced graph $\hat{\mathbb{I}}$, and edge cloud environment $\mathbf{G}$ as the input. The output is the microservices deployment strategy $\mathbf{X}$ and the makespan $\mathbf{T}$. Firstly, we determine whether the total storage of all edge servers can accommodate all microservices in the enhanced graph $\hat{\mathbb{I}}$ same as algirhtm1 lines 1 to 4. If $\mathbf{G}$ can accommodate all microservices, where $sum_\phi \geq sum_m$, we sort servers in descending order of computing capability $c_{(v_k)}$ in line 3, and sort microservices in descending order of required computing capability q_{m_i} in line 4. Then, we deploy the microservices sequentially in lines 5 to 13. For each v_k in sorted V, we start the deployment by checking whether v_k has available storage resources in line 7. If the current remaining resources $\phi_{v_k} > 0$, we place m_i^h on the server v_k in line 8 and update ϕ_{v_k}. Based on that, we update the deployment list $\mathbf{X}$ and calculate the makespan $\mathbf{T}$ in lines 10 to 11. If v_k has no further storage resources, we update set V by removing v_k, i.e., $V = \{V/v_k\}$, and then we go back to line 5. Finally, the deployment strategy $\mathbf{X}$ and the makespan $\mathbf{T}$ are returned in line 16.

***Theorem** 2:* OMDD-UB is an optimal solution on solving **P1** under the constraints (7)-(8), (10).

Proof: We prove this theorem by contradiction. We assume that deploying microservices with higher computational requirements on a server with lower computing capability denoted as $\mathbf{T}'$, will require a shorter period of time for completion than deploying microservices with higher computational requirements on a server with higher computing capability, denoted as $\mathbf{T}''$, i.e., $\mathbf{T}' < \mathbf{T}''$. Suppose

Algorithm 2 OMDD with unlimited-bandwidth (OMDD-UB)

Require: $\hat{\mathbb{I}}$, **G**.
Ensure: **X**, **T**.
1: Same as Algorithm 1 in lines 1-4;
2: **if** $sum_\phi \geq sum_m$ **then**
3: $V \leftarrow$ Update with order by $v_k = \arg\max\{c_{(v_k)}\}$;
4: $\hat{\mathbb{I}} \leftarrow$ Update with order by $\hat{\mathbb{I}} = \arg\max\{q_{m_i}\}$;
5: **for** each m_i in $\hat{\mathbb{I}}$ **do**
6: **for** $v_k \in V$ **do**
7: **if** $\phi_{v_k} > 0$ **then**
8: Place m_i on edge server v_k;
9: $\phi_{v_k} = \phi_{v_k} - 1$;
10: Update the deployment list **X**;
11: $T \mathrel{+}= q_{m_i}/c_{(v_i)}$;
12: **else**
13: Update $V = V - v_k$ and go back to line 5;
14: **else**
15: **Break**
16: **Return X**, **T**

there are two edge servers v_1 and v_2, where $c_{(v_1)} < c_{(v_2)}$. We consider the case of service with two microservices m_x and m_y, where $q_{m_y} > q_{m_x}$. When microservice with higher requiring computational capacity on a server with lower computing capability, m_x is located on edge server v_1, and m_y is located on edge server v_2. Given this, the completion time $\mathbf{T}'$ for separate deployment is: $\mathbf{T}' = q_{m_x}/c_{(v_1)} + q_{m_y}/c_{(v_2)} + r_{m_x \to m_y}/b_{v_1,v_2}$. Since the bandwidth is unlimited, communication time can be neglected. Therefore, $\mathbf{T}' = q_{m_x}/c_{(v_1)} + q_{m_y}/c_{(v_2)}$. Then, we consider the case that placing microservices with higher requiring computational capacity on a server with higher computing capability, which has $\mathbf{T}'' = q_{m_x}/c_{(v_2)} + q_{m_y}/c_{(v_1)}$. We calculate the difference between $\mathbf{T}'$ and $\mathbf{T}''$, where $\mathbf{T}' - \mathbf{T}'' = (q_{m_x}/c_{(v_1)}) + (q_{m_y}/c_{(v_2)}) - (q_{m_x}/c_{(v_2)}) - (q_{m_y}/c_{(v_1)}) = (c_{(v_2)}q_{m_x} + c_{(v_1)}q_{m_y} - c_{(v_1)}q_{m_x} - c_{(v_2)}q_{m_y})c_{(v_1)}c_{(v_2)} = (c_{(v_2)} - c_{(v_1)})(q_{m_x} - q_{m_y})/c_{(v_1)}c_{(v_2)}$. Since we suppose $c_{(v_2)} > c_{(v_1)}$ and $q_{m_x} > q_{m_y}$, we have $\mathbf{T}' - \mathbf{T}'' > 0$, i.e., $\mathbf{T}' > \mathbf{T}''$, which contradicts our assumption. Therefore, we can obtain that OMDD-UB can minimize **P1** under the constraints (7)-(8), (10). ■

D. Scenario 3: OMDD with constraints (7)-(10)

In this scenario, we investigate a more complicated scenario with all resource constraints on storage, computation, and communication, which is OMDD with constraints (7)-(10). In order to reduce the complexity of the problem, we introduce a novel definition of the main path for microservices as follows.

***Definition** 2 (main path):* The main path p_i refers to the path with the maximum weight $\arg\max\{w_{(p_i)}\}$ of $\hat{\mathbb{I}}$.

Here, we use $P = \{p_i\}$ to denote the set of all simple paths of $\hat{\mathbb{I}}$, and we use p_i to denote a simple path. Here, suppose a simple path p_i consists of a series of microservices denoted as $S^{(p_i)} = \{m_1, m_2, \ldots, m_n\}$. We treat the computation demand q_{m_i} of each microservice as the node weight and the data flow size $r_{m_i \to m_j}$ between microservices as the edge weight. Thus, for any path p_i, the weight $w_{(p_i)}$ is calculated as:

$$w_{(p_i)} = \sum_{i=1}^{n} q_{m_i} + \sum_{i=1}^{n-1} r_{m_i \to m_{i+1}}, \quad m_i \in S^{(p_i)}. \tag{12}$$

We use $t_{(p_i)}$ to represent the completion time of path p_i. The formula to calculate $t_{(p_i)}$ is as follows:

$$t_{(p_i)} = d_c(1) + \sum_{j=2}^{n} \left(d_c(j) + d_l(j-1, j)\right), \tag{13}$$

and the makespan $\mathbf{T}$ can be transformed into the maximum value of all path completion times $t_{(p_i)}$, where

$$\mathbf{T} = \max\left\{f_i(j)\right\} \equiv \max\left(t_{(p_i)}\right). \tag{14}$$

On the basis of this, we propose a preliminary deployment strategy by introducing a main path embedding method in Algorithm 3. We use the enhanced graph $\hat{\mathbb{I}}$ and edge network **G** as the input. The output is the preliminary microservices deployment strategy $\mathbf{X}_0$ and the makespan $\mathbf{T}_0$. Firstly, we determine whether the total storage of all edge servers can accommodate all microservices in an enhanced graph $\hat{\mathbb{I}}$ same as Algorithm 1 lines 1 to 4. When the edge network **G** can not accommodate all microservices where $sum_\phi < sum_m$, then break. Otherwise, we need to place these microservices separately on different servers. In order to optimize both computation and transferring time, we propose a new definition of the preferred server as follows.

***Definition** 3 (preferred server):* Let v° indicate the preferred server of V, where $v^\circ = \max_{\xi(v_k)}\{v_k|_{v_k \in V}\}$. Here, $\xi(v_k)$ is the priority value of v_k with the sum of the computing capacity and the maximum bandwidth that is connected in **G**.

Here, we define function $\xi(v_k)$ to calculate the priority value in order to find the preferred server in lines 5 to 6, aiming to obtain a server with better processing capacity and bandwidth, and jointly optimize the processing time and transferring time. For each v_k in set V, we calculate the sum of computing capability $c_{(v_k)}$ and the maximum bandwidth connected by v_k. We choose the server with the largest $\xi(v_k)$ value as a preferred server in line 7. We use the depth-first search to find all simple paths of set $P = \{p_i\}$ in line 8. Then, we deploy the microservices in lines 9 to 19. Lines 9 to 10 use equation (11) to calculate the weight of each path, and then find the main path p_i. Then, we need to determine whether the v° storage is sufficient to accommodate all the microservices of p_i, where $v^\circ \leq p_i$. We place all microservices on the preferred server when the v° storage is sufficient in line 12. Otherwise, we divide the path based on the maximum cut c_{p_i} in line 14 which is defined as follows.

***Definition** 4 (maximum cut):* Let c_{p_i} indicate the maximum cut of path p_i in $\hat{\mathbb{I}}$ which constructs by $|\phi_{v^\circ}|$ microservices with the largest weights combination.

Here, $|\phi_{v^\circ}|$ represents the server storage of the preferred server

Algorithm 3 OMDD based on Main Path Embedding (OMDD-MPE)

Require: $\hat{\mathbb{I}}$, **G**.
Ensure: $\mathbf{X}_0$, $\mathbf{T}_0$.
1: Same as Algorithm 1 in lines 1-4;
2: **if** $sum_\phi < sum_m$ **then**
3: **break**;
4: **else**
5: **for** $v_k \leftarrow 1$ **to** $|V|$ **do**
6: $\xi(v_k) = c_{(v_k)} + \max\{b_{(v_k,v_q)}\}$;
7: Choose the preferred server v°;
8: Construct set P by depth-first search;
9: **for** p_i in P **do**
10: Find the main path p_i by Equation (12);
11: **if** $\phi_{v^\circ} > |p_i|$ **then**
12: Place p_i on v°.
13: **else**
14: Construct c_{p_i} with $|\phi_{v^\circ}|$ microservices;
15: Place c_{p_i} on v°;
16: Update $p_i = p_i - c_{p_i}$ and go back to line 11;
17: Update the deployment list $\mathbf{X}_0$;
18: Update t_{p_i} by equation (12);
19: Update $\mathbf{T}_0$ by equation (13);
20: **Return** $\mathbf{X}_0$, $\mathbf{T}_0$;

v°. Then, we deploy the maximum cut c_{p_i} on v° in line 15. We update path p_i with $p_i = p_i - c_{p_i}$ to continue to complete the deployment of the remaining services and go back to line 11. After that, we update the deployment list $\mathbf{X}_0$, and we use equations (13) and (14) to calculate the makespan $\mathbf{T}_0$ in lines 17 to 19. Finally, the microservices deployment strategy $\mathbf{X}_0$ and the makespan $\mathbf{T}_0$ are returned in line 20.

Then, we utilize the preliminary deployment solution $\mathbf{X}_0$ obtained from Algorithm 3 as a starting point and introduce a novel strategy based on the improved simulated annealing algorithm for iterative optimization. However, due to the limitation of server storage resources, the traditional simulated annealing algorithm might exceed the capacity constraint when searching for neighbor solutions. To address this issue, we have refined the algorithm to overcome this challenge. The specific steps are presented in Algorithm 4. In line 1, we take the preliminary deployment strategy $\mathbf{X}_0$ and makespan $\mathbf{T}_0$ obtained in Algorithm 3 as the required values of Algorithm 4, and we set the values of hyperparameters. Lines 3 to 4 randomly select the deployment positions of two microservices from the current solution for exchanging and generating an updating solution. On the basis of that, we determine an updating makespan $\hat{\mathbf{T}}$. Lines 5 to 10 determine whether the new solution is accepted based on the Metropolis criterion. If the makespan of the updating strategy $\hat{\mathbf{T}}$ is lower than the preliminary one $\mathbf{T}$, we accept the updating strategy $\hat{\mathbf{X}}$ in line 6. Otherwise, calculate the probability ρ of the new strategy in

Algorithm 4 OMDD based on Improved Simulated Annealing (OMDD-ISA)

Require: $\hat{\mathbb{I}}$, **G**, $\mathbf{X}_0$, $\mathbf{T}_0$.
Ensure: **X**, **T**.
1: Initialize $\mathbf{X} \leftarrow \mathbf{X}_0$, $\mathbf{T} \leftarrow \mathbf{T}_0$, r, t, k ▷ r controls the speed of cooling, t is the temperature of the system, k is the number of iterations.
2: **for** $i \leftarrow 1$ to k **do**
3: Exchange the deployment positions of microservices in $\mathbf{X}$ and generate an updating deployment $\hat{\mathbf{X}}$;
4: Calculate makespan $\hat{\mathbf{T}}$ of $\hat{\mathbf{X}}$.
5: **if** $\hat{\mathbf{T}} < \mathbf{T}$ **then**
6: $\mathbf{X} \leftarrow \hat{\mathbf{X}}$
7: **else**
8: Calculate $\rho = e^{(\mathbf{T}-\hat{\mathbf{T}})/t}$
9: **if** $\varepsilon < \rho$ **then**
10: $\mathbf{X} \leftarrow \hat{\mathbf{X}}$
11: $t \leftarrow t \times r$
12: **Return** **X**, **T**

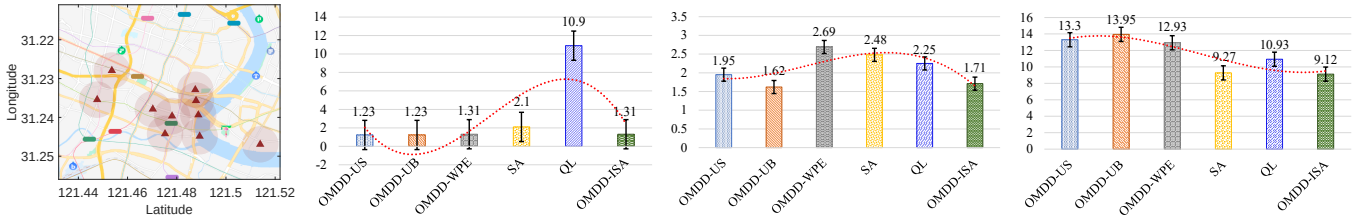
line 8. After that, we use ε to represent the judging condition for accepting ρ which is a random probability between 0 and 1, where $\varepsilon \in [0, 1]$. The new approach is acceptable if the probability ρ is above ε. Otherwise, reserve strategy $\mathbf{X}$. After that, we cool down the t at a rate r range from [0.9,1) in line 11. This optimization strategy enables us to progressively modify the distribution of microservices depending on the preliminary solution to better respond to the practical environment and resource constraints. Through multiple iterations, the simulated annealing algorithm progressively converges towards improved solutions, thereby enhancing the quality and effectiveness of the deployment strategy.

V. Experiment

A. Basic Setting

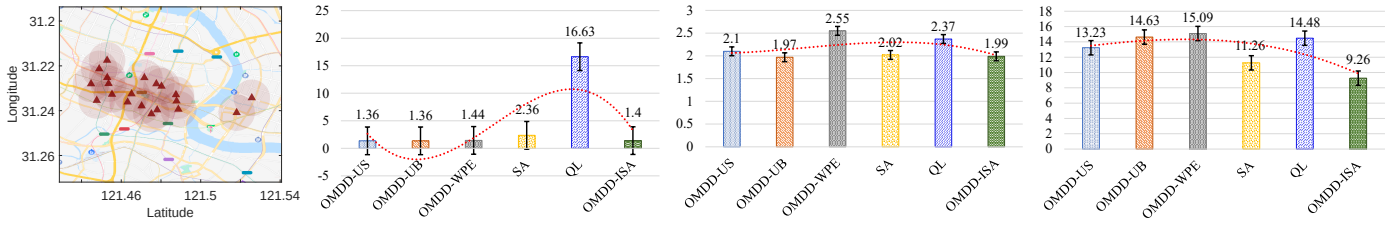
We conducted extensive experiments to validate the effectiveness of our algorithms under various scenarios. All experiments were conducted using Python 3.7 on Windows 10 with an Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz, NVIDIA RTX5000 GPU, and 32GB memory. We utilized a dataset obtained from China Telecom Shanghai Company [14], containing information about 3,233 base station locations and their corresponding user connections in June 2014. We randomly selected subsets of locations containing 6, 20, and 50 base stations, and each base station was equipped with a server, forming set V. We set the required processing capacities and internal data flows for microservices. In addition, we have made modifications to the edge network in accordance with various scenarios.

1) Scenario 1: We set computing capacities of edge servers to range from $[5, 20]$ $gflops$, and storage resources range from $[120, 150]$ units. We set the inter-server bandwidth to range from $[20, 80]$ GB/s. Additionally, the required computing



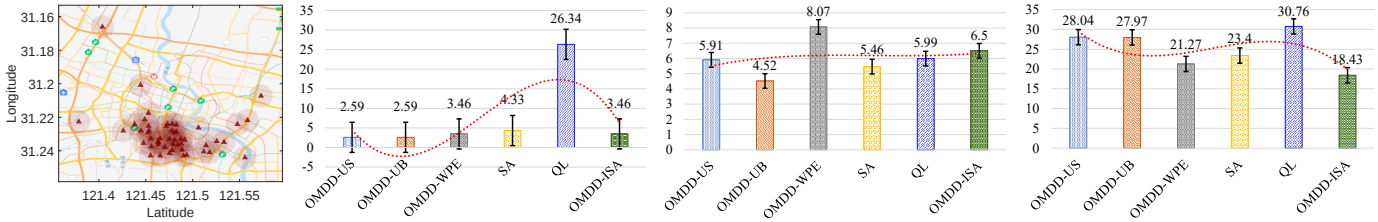
(a) Server locations (# of 10). (b) Scenario 1. (c) Scenario 2. (d) Scenario 3.

Fig. 2. 10 servers and 25 microservices.



(a) Server locations (# of 20). (b) Scenario 1. (c) Scenario 2. (d) Scenario 3.

Fig. 3. 20 servers and 50 microservices.



(a) Server locations (# of 50). (b) Scenario 1. (c) Scenario 2. (d) Scenario 3.

Fig. 4. 50 servers and 120 microservices.

capacities of microservices range from $[1, 3]$ $gflops$, and the inter-microservice data flow sizes range from $[1, 80]$ GB/s.

2) Scenario 2: We set computing capacities of servers to range from $[5, 20]$ $gflops$, and storage resources range from $[1, 5]$ units. We set the inter-server bandwidth to range from $[100, 500]$ GB/s. The setting of required computing capacities for microservices is the same as scenario 1, and the inter-microservice data flow sizes are changed to $[0.1, 0.2]$ GB/s.

3) Scenario 3: We set storage resources range from $[1, 5]$ units. In addition, the computing capacities, inter-server bandwidth ranges, required computing capacities, and inter-microservice data flow size ranges are similar to scenario 1.

For scenarios 1 and 2, we have already demonstrated that OMDD-US and OMDD-UB are optimal in Section IV. For scenario 3, we introduced four baselines (OMDD-US, OMDD-UB, Simulated Annealing-only, Q-Learning) in comparison with our proposed OMDD-ISA algorithm:

- **Simulated Annealing-only (SA)**: Traditional annealing algorithm, generating random initial values.
- **Q-Learning (QL)**: States are composed of the allocation status of a series of services. Each service can be assigned to different servers or remain unassigned, and the action space contains $\sum_{h=1}^{|S|} |M_h| * |V|$ actions.

B. Experiment Results

We evaluated different algorithms in various scenarios to compare their performance on microservices applications of different scales and scenarios. The experimental results indicate that in Scenario 1, as shown in Figures 2(b), 3(b), and 4(b), the OMDD-US method consistently minimizes the makespan. Notably, the makespan of OMDD-UB is consistently the same as OMDD-US because both methods greedily place microservices on high-computing-capacity servers. The difference is that OMDD-UB sorts microservices based on $q_{m_i^h}$ before placement, making it more complex in large-scale scenarios. Meanwhile, OMDD-MPE and OMDD-ISA perform relatively modestly in Scenario 1, with slightly higher makespans compared to other methods. This is because OMDD-MPE takes into account both computation and communication resources. In Scenario 1, where storage resources are unrestricted, placing all services on a single server does not generate communication time, making communication resources less critical. Furthermore, the SA method exhibits relatively poorer performance in Scenario 1 due to its tendency to get trapped in local optimal when storage resources are unlimited and the problem scale is substantial. Finally, the QL method shows relatively higher makespans across all problem scales in Scenario 1. This is attributed to the complexity of its state space, resulting in longer computation times and the inability to find the optimal solution within limited iterations.

In Scenario 2, as depicted in Figures 2(c), 3(c), and 4(c), the OMDD-UB method excels with the lowest makespan. The OMDD-US method performs relatively well in Scenario 2,

though with a slightly higher makespan than OMDD-UB. This is due to OMDD-US not fully utilizing the computation capacity of edge servers. Some microservices requiring lesser computing resources are deployed on powerful servers, leading to resource wastage and higher time. OMDD-MPE performs less favorably in Scenario 2 due to its consideration of both computation and communication resources, where communication resources are not the primary bottleneck. The SA and QL methods demonstrate moderate performance in Scenario 2, with relatively high makespans across different problem scales, possibly due to their stochastic nature causing significant fluctuations. In Scenario 2, OMDD-ISA runs effectively with a lower makespan. The main reason it falls short of OMDD-UB is due to its stochastic character, which could make it impossible to find the optimal solutions.

In Scenario 3, as shown in Figures 2(d), 3(d), and 4(d), we observe that OMDD-ISA consistently achieves a lower makespan across all scenarios, showcasing the significant optimization achieved by introducing the simulated annealing algorithm. Comparing the algorithm performance across different scenarios, we observe the following trends: in small-scale environments, the difference between SA and OMDD-ISA is not significant, primarily because of fewer placement strategies available at smaller scales, leading to reasonable solutions for both methods. In larger-scale environments, the SA algorithm's performance diminishes, possibly due to the randomness of the initial solution leading to local optimal. Additionally, QL solutions exhibit fluctuation, with deteriorating effects as the scale increases. This is attributed to the vast state space in QL in large-scale environments, making it difficult to find global optima within limited iterations. In conclusion, the experimental results effectively demonstrate the effectiveness and superiority of the proposed algorithms in various scenarios.

VI. Conclusion

This paper focuses on addressing the microservice deployment with dependencies in a resource-constrained mobile edge computing environment. We explore how to optimize the deployment of microservices in various application scenarios. We initially consider two straightforward scenarios: one with unlimited storage resources under the bandwidth constraint, and the other with unlimited bandwidth resources under the storage constraint. For each of these two scenarios, we introduce a novel enhanced graph construction method and design two optimal solutions. For Scenario 3, which involves complex constraints on server capacity, computational capability, and communication resources, we present an optimization method based on main path partitioning and the simulated annealing algorithm. By flexibly exploring the solution space, we incrementally optimize microservices deployment to adapt to real-world environments and resource constraints. Across multiple experimental results, our approach significantly improves microservice deployment efficiency and overall performance compared to baseline strategies.

Acknowledgment

This work is supported by the Beijing Natural Science Foundation (4192007), and supported by the National Natural Science Foundation under grant (92267107) along with other government sponsors.

References

[1] Ding Z, Wang S, Jiang C. Kubernetes-oriented microservice placement with dynamic resource allocation[J]. IEEE Transactions on Cloud Computing, 2022.

[2] Tang B, Guo F, Cao B, et al. Cost-aware Deployment of Microservices for IoT Applications in Mobile Edge Computing Environment[J]. IEEE Transactions on Network and Service Management, 2022.

[3] Carrusca A, Gomes M C, Leitão J. Microservices management on cloud/edge environments[C]//Service-Oriented Computing–ICSOC 2019 Workshops: WESOACS, ASOCA, ISYCC, TBCE, and STRAPS, Toulouse, France, October 28–31, 2019, Revised Selected Papers 17. Springer International Publishing, 2020: 95-108.

[4] Wang S, Guo Y, Zhang N, et al. Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach[J]. IEEE Transactions on Mobile Computing, 2019, 20(3): 939-951.

[5] Menouer, T., Khedimi, A., Cérin, C. et al. Cloud-Native Placement Strategies of Service Function Chains with Dependencies. J Netw Syst Manage 31, 47 (2023).

[6] Zhao H, Deng S, Liu Z, et al. Distributed redundant placement for microservice-based applications at the edge[J]. IEEE Transactions on Services Computing, 2020, 15(3): 1732-1745.

[7] Guerrero C, Lera I, Juiz C. Resource optimization of container orchestration: a case study in multi-cloud microservices-based applications[J]. The Journal of Supercomputing, 2018, 74(7): 2956-2983.

[8] Deng S, Zhao H, Xiang Z, et al. Dependent function embedding for distributed serverless edge computing[J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 33(10): 2346-2357.

[9] Khare S, Sun H, Gascon-Samson J, et al. Linearize, predict and place: minimizing the makespan for edge-based stream processing of directed acyclic graphs[C]//Proceedings of the 4th ACM/IEEE Symposium on Edge Computing. 2019: 1-14.

[10] Pallewatta S, Kostakos V, Buyya R. QoS-aware placement of microservices-based IoT applications in Fog computing environments[J]. Future Generation Computer Systems, 2022, 131: 121-136.

[11] Liao H, Li X, Guo D, et al. Dependency-aware application assigning and scheduling in edge computing[J]. IEEE Internet of Things Journal, 2021, 9(6): 4451-4463.

[12] Cerny T, Abdelfattah A S, Bushong V, et al. Microservice architecture reconstruction and visualization techniques: A review[C]//2022 IEEE International Conference on Service-Oriented System Engineering (SOSE). IEEE, 2022: 39-48.

[13] Guo Y, Wang S, Zhou A, et al. User allocation-aware edge cloud placement in mobile edge computing[J]. Software: Practice and Experience, 2020, 50(5): 489-502.

[14] Li Y, Zhou A, Ma X, et al. Profit-aware edge server placement[J]. IEEE Internet of Things Journal, 2021, 9(1): 55-67.

[15] Pinedo M L. Scheduling[M]. New York: Springer, 2012.

[16] Duc T L, Leiva R G, Casari P, et al. Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey[J]. ACM Computing Surveys (CSUR), 2019, 52(5): 1-39.

[17] Oakes E, Yang L, Zhou D, et al. SOCK: Rapid task provisioning with Serverless-Optimized containers[C]//2018 USENIX annual technical conference (USENIX ATC 18). 2018: 57-70.

[18] Niu Y, Liu F, Li Z. Load balancing across microservices[C]//IEEE INFOCOM 2018-IEEE Conference on Computer Communications. IEEE, 2018: 198-206.

[19] Lv W, Yang P, Zheng T, et al. Graph Reinforcement Learning-based Dependency-Aware Microservice Deployment in Edge Computing[J]. IEEE Internet of Things Journal, 2023.

[20] Lv W, Wang Q, Yang P, et al. Microservice deployment in edge computing based on deep Q learning[J]. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(11): 2968-2978.