# How to Share: Balancing Layer and Chain Sharing in Industrial Microservice Deployment

Yuxiang Liu, Bo Yang, Senior Member, IEEE, Yu Wu, Student Member, IEEE, Cailian Chen, Member, IEEE, and Xinping Guan, Fellow, IEEE

Abstract—With the rapid development of smart manufacturing, edge computing-oriented microservice platforms are emerging as an important part of production control. In the containerized deployment of microservices, layer sharing can reduce the huge bandwidth consumption caused by image pulling, and chain sharing can reduce communication overhead caused by communication between microservices. The two sharing methods use the characteristics of each microservice to share resources during deployment. However, due to the limited resources of edge servers, it is difficult to meet the optimization goals of the two methods at the same time. Therefore, it is of critical importance to realize the improvement of service response efficiency by balancing the two sharing methods. This paper studies the optimal microservice deployment strategy that can balance layer sharing and chain sharing of microservices. We build a problem that minimizes microservice image pull delay and communication overhead and transform the problem into a linearly constrained integer quadratic programming problem through model reconstruction. A deployment strategy is obtained through the successive convex approximation (SCA) method. Experimental results show that the proposed deployment strategy can balance the two resource sharing methods. When the two sharing methods are equally considered, the average image pull delay can be reduced to 65% of the baseline, and the average communication overhead can be reduced to 30% of the baseline.

Index Terms—Industrial Internet of Things (IIoT), microservice deployment, layer sharing, chain sharing.

# 1 INTRODUCTION

W ITH the rapid development of smart manufacturing and flexible production, the flexibility of industrial production has been greatly enhanced [1], [2]. Industrial software needs to quickly redistribute and adjust production processes according to changes of orders, which has higher requirements for flexibility and scalability of industrial software [3], [4], [5]. Traditional industrial software adopts a monolithic service architecture. The high coupling and occupancy rate within the service will increase the complexity of the whole system. Its scalability, stability, and fault tolerance are difficult to meet the requirements of smart manufacturing. Therefore, the industrial software architecture based on microservices has been widely concerned [6], [7]. Through the microservice architecture, a complete service can be split into multiple loosely coupled microservices. Different microservices are logically independent and have a high degree of flexibility, scalability, and fault tolerance, which can well adapt to the requirements of smart manufacturing.

To meet the high requirements of computation-intensive tasks for real-time performance and service efficiency in smart manufacturing, edge computing-oriented microservice platforms are emerging [8], [9], [10], [11], [12]. At present, container technologies represented by Docker [13] and container orchestration tools represented by Kubernetes [14] are becoming mainstream solutions for microservice deployment and maintenance on edge platforms. According to different service requests and deployment strategies, each microservice which is packaged into a Docker image can be deployed to edge servers through container orchestration tools.

In the containerized deployment of microservices, service efficiency is an important indicator for evaluating the quality of the deployment solution. Service efficiency is mainly affected by two aspects. One is the startup time of microservices. It mainly depends on the pull delay of Docker images which are stored in the cloud through different image layers [15]. When a microservice needs to be provided locally, the edge server will pull a nonlocal container image containing all required layers from the cloud. Due to limited network bandwidth, image pulls incur a corresponding downlink delay. A comprehensive research shows that with a bandwidth of 100Mbps, the average startup time of a single image is about 20.7 seconds, while the average image pull delay is about 15.8 seconds, accounting for 76.6% of the average startup time [16]. Image pull delay has become a non-negligible factor affecting container startup time, which in turn affects the efficiency of service response. The other is the communication overhead between microservices. It depends on the amount of data communicated between microservices. An industrial application can be completed by multiple microservices deployed on one or more edge servers [17]. These microservices can be called microservice chains, and there will be frequent data exchanges between microservices in the same microservice chain [18]. A large amount of data transmission between microservices will cause high transmission delay, which will affect the service response efficiency.

Y. Liu, B. Yang (Corresponding author), Y. Wu, C. Chen, and X. Guan are with the Department of Automation, Shanghai Jiao Tong University, Shanghai 200240, China; Key Laboratory of System Control and Information Processing, Ministry of Education of China, Shanghai 200240, China; Shanghai Engineering Research Center of Intelligent Control and Management, Shanghai 200240, China (e-mail: liu953973860@sjtu.edu.cn; bo.yang@sjtu.edu.cn; 5wuuy5@sjtu.edu.cn; cailianchen@sjtu.edu.cn; xpguan@sjtu.edu.cn).

Due to the above two aspects, it is very important to improve service efficiency through resource sharing. There are two types of resource sharing strategies for the improvement of service efficiency. One of the strategies for resource sharing is layer sharing [15]. Docker natively supports the sharing of layers. If the microservices deployed on the same edge server use the same image layer, the layer will not be pulled repeatedly when pulling images. This layer can be shared by all microservices on the server. The image pull delay can be effectively reduced by layer sharing, thereby improving the startup speed and service response efficiency of microservices. The other strategy for resource sharing is chain sharing [18], [19], which can be defined as the data sharing of microservices deployed on the same server. In the microservice chain, there is frequent data transfer between two adjacent microservices. If two microservices are deployed on the same server, the data can be directly accessed through chain sharing by the next microservice without multi-hop transmission of data. The delay and packet loss caused by data transmission can be reduced by chain sharing.

However, due to the limited resources of edge servers, it is impossible for all microservices to be deployed on the same edge server. Therefore, it is necessary to find an optimal microservice deployment strategy for the trade-off between layer sharing and chain sharing. Besides service efficiency, due to the limited resources of edge servers, the microservice deployment strategy can not make full use of different resources (such as computing and storage resources) at the same time, resulting in idle computing resources. Therefore, a method is also needed to reasonably allocate resources to different microservices deployed on a server and maximize the utilization of resources.

Aiming at resource sharing and maximizing resource utilization problems among microservices, the deployment of microservices mainly faces the following difficulties. 1) How to model the layered structure of the microservice image to accurately describe the relationship between the microservice image and the container layer. 2) How to describe the chain structure of microservices and the communication between microservices. 3) How to balance layer sharing and chain sharing to establish an optimization problem to achieve the best deployment strategy. 4) How to reallocate resources to microservices deployed on edge servers to make full use of computing resources. In this paper, we study the microservice deployment problem considering microservice layer sharing and chain sharing. The problem is modeled as an integer programming problem that minimizes image pull delay and communication overhead. Based on this problem, a microservice deployment strategy and resource redistribution scheme are proposed. The main contributions of this work are as follows:

- 1) We describe the layered structure and chain structure of microservices through the same model. An integer programming problem is established to minimize the image pull delay and communication overhead.
- 2) Through model reconstruction, we prove that the integer programming problem can be transformed into an integer quadratic programming problem with linear constraints. The optimal solution is obtained by using the successive convex approximation (SCA) method. This

method can effectively balance the image pull delay and communication overhead.

- 3) A resource redistribution algorithm for edge servers is proposed to make full use of idle computing resources.
- 4) Through experiments, the results are evaluated in multiple dimensions, such as image pull delay and interservice communication overhead. These experiments demonstrate the effectiveness of the proposed method.

The remainder of this paper is organized as follows. Sec. 2 briefly reviews the related literature. In Sec. 3, the layered structure and chain structure of the system are modeled, and the problem formulation is given. Sec. 4 solves the proposed problem. Sec. 5 proposes a resource redistribution algorithm for edge servers. Sec. 6 evaluates the results of the proposed method. Sec. 7 discusses the limitations and future work. Sec. 8 concludes the paper.

## 2 RELATED WORKS

In this section, we discuss current research on the deployment of microservices.

In recent years, optimizing the cost and improving microservice response efficiency have received wide attention. Herrera et al. [20] designed a distributed microservice deployment framework named DADO to optimize the response time of microservices. Deng et al. [21] and Chen et al. [22] proposed algorithms to solve the cost-aware microservice deployment problem, they considered application deployment cost and service migration cost, respectively. Fadda et al. [23] provided an approach for supporting the deployment of microservices in multi-cloud environments to optimize the quality and cost. Zhao et al. [24] developed a cost-aware elastic microservice deployment algorithm to solve the container-based microservice deployment problem. The above researchers have conducted sufficient research on service response efficiency and service quality. However, these studies do not consider the characteristics of microservices, such as the chain structure of multiple microservices and the layered structure due to containerized deployment.

From the perspective of the chain structure, deployment strategies become more complex due to the dependencies between microservices. In general, a microservice chain can be modeled as a directed acyclic graph [25], [26]. Wang et al. [17] and Li et al. [27] proposed algorithms to solve latencyaware microservice deployment problems. Armani et al. [28] proposed a cost-effective workload distribution strategy for microservice-based applications considering fault tolerance and load balancing of microservice chains. Sasabe et al. [29] considered the service chaining and function placement problems to optimize the total delay. Lv et al. [18] considered the containerized deployment of microservices, and a chain sharing deployment strategy is proposed to minimize the communication overhead. The above research focuses on the chain structure and chain sharing of microservices, but it does not take into account the layered structure in containerized deployments of microservices.

For the layered structure, researchers have focused on how to reduce the latency of image pulling by reducing image size or utilizing layer sharing of images. Lou *et al.* [30] considered layer sharing among images and proposed



Fig. 1: An example of layer sharing and chain sharing deployment strategies

a layer-aware scheduling algorithm. Gu *et al.* [31] designed a microservice deployment and request scheduling strategy based on layer sharing and used an iterative greedy algorithm to obtain the optimal strategy to improve the throughput of microservices. Gu *et al.* [32] also investigated the problem of how to collaboratively deploy microservices by incorporating both intra-server and inter-server layer sharing to maximize the edge throughput. The above research fully considers the layer sharing strategy in the containerized deployment of microservices, but does not consider the sharing strategy in the presence of the chain structure between microservices.

Although the existing schemes have considered optimizing the efficiency and cost of microservice deployment in terms of the layered structure and chain structure, respectively, there are still some challenging problems to be solved. First, in complex intelligent manufacturing scenarios, the layered structure and chain structure usually coexist. In this case, the deployment strategies of layer sharing and chain sharing will affect each other, which further increases the difficulty of finding an effective deployment strategy. Secondly, an uneven deployment strategy will lead to idle server resources due to inconsistent server resources required by different microservices. In order to solve these problems, we propose a microservice deployment scheme that comprehensively considers layer sharing and chain sharing. It can both reduce the delay of image pulling and the communication overhead. The idle server resources can also be fully utilized. Our scheme can effectively improve the operating efficiency of microservices. To the best of our knowledge, there is no research that comprehensively considers the two sharing methods.

# **3** SYSTEM MODELING AND PROBLEM FORMULA-TION

#### 3.1 A simple example

First, we show the two strategies of layer sharing and chain sharing through a simple microservice deployment model. As shown in Fig. 1, we consider two applications composed of two and three microservices, respectively. We denote the *i*th microservice in application k as  $ms^{ki}$ . Each microservice image consists of a different number of image layers. The bandwidth between the three servers and the cloud server is 120 MB/s, and the two adjacent servers can be reached with a single hop. Under the layer sharing deployment strategy, the same image layer on the same edge server can be shared. Therefore, the size of the image layer to be pulled is 1617.48 MB, and the total download time is 13.479 seconds. However, the total communication data is 1031 KB because of communication between servers. Under the chain sharing deployment strategy, the total communication data is 0 KB since the microservices in the same chain are all deployed on the same server. The size of the image layer to be pulled is 2283 MB because there is no layer sharing, and the total pull delay is 19.025 seconds. When considering both chain sharing and layer sharing, we can get the result shown in Fig. 1(c). The size of the image layer to be pulled is 1758 MB, the total download time is 14.65 seconds, and the total communication data is 315 KB. These data can be found in Table 1. It can be seen that different deployment strategies have a significant impact on image pull delay and communication overhead. If layer sharing and chain sharing can be both considered, we will get low image pull delay and low communication overhead at the same time.

TABLE 1: Comparison about layer sharing and chain sharing

	image pull delay	communication overhead			
layer sharing	13.479s	1031 KB			
chain sharing	19.025s	0 KB			
both	14.65s	315 KB			

#### 3.2 System model

We consider an intelligent manufacturing system, which has M production devices and N edge servers. We define device set as  $\mathbb{M} = \{1, 2, \cdots, M\}$  and server set as  $\mathbb{N} = \{1, 2, \cdots, N\}$ . A cloud server is deployed at the remote end to store the microservice images. Each production device is connected to the nearest edge server. Each edge server has limited computing and storage resources, and a certain number of microservices can be deployed on it. The computing and storage resources of edge server n are denoted as  $C_n^C$  and  $C_n^S$ , and the bandwidth between cloud server and edge server n is  $b_n^{cloud}$ .

Suppose there are several industrial applications, and application set is defined as  $\mathbb{K} = \{1, 2, \dots, K\}$ . Each application is composed of multiple microservices. The microservice set in the *k*th application is  $\mathbb{A}_k = \{1, 2, \dots, A_k\}$ , where  $A_k$  is the amount of microservices in the *k*th application. Each application can handle service requests from production device. We use  $ms^{ki}$  to denote the *i*th microservice in application k, and  $u^{ki}$  to denote the computing resources requested by microservice  $ms^{ki}$ .

All microservice images are stored in the microservice image library of the cloud server, and are pulled by the edge server according to the deployed microservices. Every microservice image consists of some shareable layers and some non-shareable layers. We use set  $\mathbb{L} = \{1, 2, \cdots, L\}$  to represent all layers of different size and  $S^l \in \mathbb{R}^+$  to represent the size of layer  $l \in \mathbb{L}$ . In this way, each microservice can be composed of one or more layers in  $\mathbb{L}$ , and  $E^{kil} \in \{0, 1\}$  can be used to indicate whether  $ms^{ki}$  contains the *l*th layer.  $E^{kil} = 1$  represents that  $ms^{ki}$  contains the *l*th layer.

Each server will receive different service requests and data. If the expected microservice is deployed on the server at this time, the service request can be processed directly. If the expected microservice is not deployed on the server, the request and data need to be transmitted to another edge server through multi-hop transmission. Due to the different geographical locations, the hops in communication between different servers is also different. We define  $D_{nn'}$  as the hops of requests or data transmitted from server n to server n', which can be obtained by the shortest communication path between the two servers. It is obvious that  $D_{nn'} = D_{n'n}$ ,  $D_{nn} = 0$ . We can use a matrix **D** to represent the multi-hop connection between all servers. The notations and variables commonly used in this paper are summarized in Table 2.

$$\mathbf{D} = \begin{bmatrix} 0 & D_{12} & D_{13} & \cdots & D_{1N} \\ D_{21} & 0 & D_{23} & \cdots & D_{2N} \\ D_{31} & D_{32} & 0 & \cdots & D_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ D_{N1} & D_{N2} & D_{N3} & \cdots & 0 \end{bmatrix}$$

Symbol	Description
<u>M M</u>	devices and the set of devices
111,111	devices and the set of devices
$N, \mathbb{N}$	servers and the set of servers
$K, \mathbb{K}$	applications and the set of applications
$L, \mathbb{L}$	layers and the set of layers
$A_k, \mathbb{A}_k$	microservices in the <i>k</i> th application and the set of microservices
$C_n^C$	the computing resources of edge server $n$
$C_n^S$	the storage resources of edge server $n$
$b_n^{cloud}$	the bandwidth between cloud server and edge server $n$
$ms^{ki}$	the $i$ th microservice in application $k$
$u^{ki}$	the computing resources requested by microservice $\boldsymbol{ms}^{ki}$
$S^l$	the size of layer <i>l</i>
$E^{kil}$	whether $ms^{ki}$ contains the <i>l</i> th layer
$D_{nn'}$	the hops of requests or data transmitted from server $n$ to server $n'$
$x_n^{ki}$	whether $ms^{ki}$ is deployed in server $n$
$d_n^l$	whether the layer $l$ is pulled to edge server $n$

#### 3.3 Problem Formulation

#### 3.3.1 Microservice deployment and layer sharing

We define  $x_n^{ki} \in \{0, 1\}$  to represent the deployment of  $ms^{ki}$ , and  $x_n^{ki} = 1$  to represent that the microservice is deployed on the edge server n, otherwise not. Due to the layered structure of microservices, once a microservice is deployed on edge server n, all layers contained in the microservice image need to be pulled to server n. We use the variable  $d_n^l \in \{0,1\}$  to represent whether the layer l is pulled to edge server n, and  $d_n^l = 1$  indicates that the lth layer needs to be downloaded to the edge server n, otherwise not.

Since each microservice can only be deployed on a unique server, we can get the following constraints:

$$\sum_{n \in \mathbb{N}} x_n^{ki} = 1, \forall k \in \mathbb{K}, \forall i \in \mathbb{A}_k$$
(1)

If microservices deployed on the same server can share the same layer, the layer only needs to be downloaded once. Therefore,  $d_n^l$  and  $x_n^{ki}$  satisfy the following constraints:

$$d_n^l = \min \{ \sum_{k \in \mathbb{K}} \sum_{i \in \mathbb{A}_k} x_n^{ki} E^{kil}, 1 \}, \forall l \in \mathbb{L}, \forall n \in \mathbb{N}$$
 (2)

Due to the limited storage resources, the layer size of the deployed microservices needs to be smaller than the storage resources of edge servers. So we can get the following constraints:

$$\sum_{l \in \mathbb{L}} d_n^l S^l \le C_n^S, \forall n \in \mathbb{N}$$
(3)

Due to the limited computing resources, the total computing resource of all microservices deployed on a server needs to be less than the computing resource of the server. So we can get the following constraints:

$$\sum_{k \in \mathbb{K}} \sum_{i \in \mathbb{A}_k} x_n^{ki} u^{ki} \le C_n^C, \forall n \in \mathbb{N}$$
(4)

For each server, all layers deployed on the server need to be pulled from the cloud. The image pull delay of server n can be expressed as follows:

$$T_n = \frac{\sum_{l \in \mathbb{L}} d_n^l S^l}{b_n^{cloud}} \tag{5}$$

#### 3.3.2 Communication overhead and chain sharing

We model the microservice chain as a directed weighted acyclic graph to reveal the impact of communication data on the deployment of microservices. Taking an application consisting of four microservices as an example, the modeled directed weighted graph is shown in Fig. 2. We use the interaction weight  $w_{ij}^k$  to represent the size of the communication traffic between each two microservices  $ms^{ki}$  and  $ms^{kj}$ .



Fig. 2: Directed weighted graph for application k

The interaction graph can be written in the form of a matrix. For application k, its interaction matrix is defined as follows:

$$\mathbf{w}^{k} = \begin{bmatrix} w_{11}^{k} & \cdots & w_{1A_{k}}^{k} \\ \vdots & \ddots & \vdots \\ w_{A_{k}1}^{k} & \cdots & w_{A_{k},A_{k}}^{k} \end{bmatrix}$$

where  $w_{ij}^k$  is non-zero value only when  $ms_i^k$  and  $ms_j^k$  are connected. For example,  $\mathbf{w}^k$  of the microservice chain shown in Fig. 2 can be defined as follows:

$$\mathbf{w}^{k} = \begin{bmatrix} 0 & w_{12}^{k} & 0 & w_{14}^{k} \\ 0 & 0 & w_{23}^{k} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Based on the interaction graph, we can calculate the communication cost. Multi-hop data transmission between two adjacent microservices is not required if they are deployed on the same edge server. To calculate the total amount of data transferred between microservices in an application, we first need to find the hops between the servers where any two microservices are deployed. For the servers where any two microservices  $ms^{ki}$  and  $ms^{kj}$  are deployed, we define  $Hop(k, i, j) = \sum_{n \in \mathbb{N}} \sum_{n' \in \mathbb{N}} x_n^{ki} x_{n'}^{kj} D_{nn'}$  to calculate the hops. For any application k, its communication overhead can be expressed as follows:

$$R^{k} = \sum_{i \in \mathbb{A}_{k}} \sum_{j \in \mathbb{A}_{k}} w_{ij}^{k} Hop(k, i, j)$$
$$= \sum_{i \in \mathbb{A}_{k}} \sum_{j \in \mathbb{A}_{k}} \left( w_{ij}^{k} \sum_{n \in N} \sum_{n' \in N} x_{n}^{ki} x_{n'}^{kj} D_{nn'} \right)$$
(6)

#### 3.3.3 The virtual microservice

Each application k originates from a service request on a production device. Each generated service request is transmitted via the network to the edge server closest to the production device at first. We define the number of the source device for the application k as  $source^k$ . For each  $source^k$ , we find its directly connected edge server  $N^k$  and define a virtual initial microservice  $ms^{k0}$  to describe the impact of request generation location on microservice deployment. We use  $ms^{k0}$  to denote the service requests generated on device  $source^k$ . The microservice set in the kth application is modified as  $\mathbb{A}_k = \{0, 1, 2, \dots, A_k\}$ . Therefore, we should add the virtual initial microservice to the interaction diagram. Fig. 2 can be modified as follows:



Fig. 3: Modified directed weighted graph for application k

The microservice  $ms^{k0}$  does not actually exist, so its required computing resource is  $u^{k0} = 0$  and does not contain any layers. When its deployment location is fixed, we can get the following constraints:

$$x_{N^k}^{k0} = 1, \forall k \in \mathbb{K}$$
(7)

# 3.3.4 Image pull delay and communication overhead minimization problem

The goal of microservice deployment is to minimize image pull delay and communication overhead under the constraints of device resources and service characteristics. The optimization problem can be expressed as follows:

P1: 
$$\min_{m \neq d} T, R$$
 (8)

s.t. 
$$(1), (2), (3), (4), (7)$$

$$x_n^{ki}, d_n^l \in \{0, 1\}$$
 (9)

where  $T = \sum_{n \in \mathbb{N}} T_n$  is the total image pull delay.  $R = \sum_{k \in \mathbb{K}} R^k$  is the total communication overhead. This problem is a multi-objective optimization problem and there is a multiplicative form of variables in  $R^k$ . Therefore, the problem is difficult to solve. In next section, we will transform the problem to a single-objective optimization problem and give a solution.

# 4 MICROSERVICE DEPLOYMENT SCHEME BASED ON SCA

## 4.1 Problem transformation

We vectorize all variables through model reconstruction to make the problem clearer and easier to solve. Then we convert all constraints to linear constraints. Finally, the problem is transformed into a single-objective integer quadratic programming problem through an additive weighted model.

#### 4.1.1 Image pull delay

Consider the first part of problem P1:

$$T = \sum_{n \in \mathbb{N}} \frac{\sum_{l \in L} d_n^l S^l}{b_n^{cloud}} \tag{10}$$

which is a linear form. We define  $\mathbf{d} = \begin{bmatrix} \mathbf{d}_1^T, \cdots, \mathbf{d}_N^T \end{bmatrix}^T$ ,  $\mathbf{S} = \begin{bmatrix} S^1, \cdots, S^L \end{bmatrix}^T$ , and  $\mathbf{M} = \begin{bmatrix} \mathbf{S}^T \\ \overline{b_1^{cloud}}, \cdots, \frac{\mathbf{S}^T}{b_N^{cloud}} \end{bmatrix}$ . Then the calculation of the total pull delay can be converted to

$$T = \mathbf{M}\mathbf{d} \tag{11}$$

#### 4.1.2 Communication overhead

Consider the second part of problem P1:

$$R = \sum_{k \in \mathbb{K}} \sum_{i \in \mathbb{A}_k} \sum_{j \in \mathbb{A}_k} \left( w_{ij}^k \sum_{n \in N} \sum_{n' \in N} x_n^{ki} x_{n'}^{kj} D_{nn'} \right)$$
(12)

We can also define  $\mathbf{x}^{ki} = [x_1^{ki}, x_2^{ki}, \cdots, x_N^{ki}]^T$ ,  $\mathbf{x}^k = [(\mathbf{x}^{k1})^T, \cdots, (\mathbf{x}^{kA_k})^T]^T$ , and  $\mathbf{x} = [(\mathbf{x}^1)^T, \cdots, (\mathbf{x}^K)^T]^T$ . We can get

$$\sum_{n \in N} \sum_{n' \in N} x_n^{ki} x_{n'}^{kj} D_{nn'} = \left( \mathbf{x}^{kj} \right)^T \mathbf{D} \mathbf{x}^{ki}$$
(13)

Let  $\odot$  be the Hadamard product of the matrix and define

$$\mathbf{W}^{k} = \mathbf{w}^{k} \odot \begin{bmatrix} \mathbf{D} & \cdots & \mathbf{D} \\ \vdots & \ddots & \vdots \\ \mathbf{D} & \cdots & \mathbf{D} \end{bmatrix}_{A_{k} \times A_{k}}$$
(14)

$$\mathbf{W} = \begin{bmatrix} \mathbf{W}^1 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \mathbf{W}^K \end{bmatrix}_{K \times K}$$
(15)

Then the calculation of the communication overhead can be converted to

$$R = \mathbf{x}^T \mathbf{W} \mathbf{x} \tag{16}$$

#### 4.1.3 Constraints

Considering equation (2), the original constraint is nonlinear. We can turn it into a linear constraint by two new constraints:

$$d_n^l \le \sum_{k \in K} \sum_{i \in A_k} x_n^{ki} E^{kil} \tag{17}$$

$$d_n^l \ge \frac{\sum_{k \in K} \sum_{i \in A_k} x_n^{ki} E^{kil}}{Z} \tag{18}$$

where Z is an arbitrarily large constant greater than 1. Equations (17) and (18) can be equivalent to constraint (2) because  $d_n^l$  is a binary variable. Therefore, all constraints in problem P1 are transformed into linear constraints.

For linear constraints, we can also vectorize all constraints in the same way as in Sec. 4.1.1, and the transformed constraints are

$$\mathbf{b}\mathbf{x} = \mathbf{b}_1 \tag{19}$$

$$\mathbf{H}\mathbf{X} = \mathbf{D}_2 \tag{20}$$

$$\mathbf{d} \leq \mathbf{Y}\mathbf{x} \tag{21}$$

$$\mathbf{d} \ge \frac{\mathbf{I} \mathbf{X}}{Z} \tag{22}$$

$$S\mathbf{d} < \mathbf{C}^S$$
 (23)

$$\mathcal{G}\mathbf{x} < \mathbf{C}^C \tag{24}$$

Constraints (19)-(24) correspond to (1), (7), (17), (18), (3), (4) respectively. Appendix A shows the detailed values of matrices Q,  $\mathbf{b}_1$ ,  $\mathbf{H}$ ,  $\mathbf{b}_2$ ,  $\mathbf{Y}$ , S,  $\mathbf{C}^S$ ,  $\mathcal{G}$ ,  $\mathbf{C}^C$ . Problem *P*1 can be transformed into

P2: 
$$\min_{x,d} T, R$$
 (25)  
s.t. (9), (19) - (24)

where  $T = \mathbf{M}\mathbf{d}$ ,  $R = \mathbf{x}^T \mathbf{W} \mathbf{x}$ .

# 4.1.4 Single objective optimization problem

The original problem has two optimization objectives. We use an additive weighting model to turn the original problem into a single-objective problem. The utility function is as follows:

$$F(\mathbf{x}, \mathbf{d}) = \theta \frac{T - T_{min}}{T_{max} - T_{min}} + (1 - \theta) \frac{R - R_{min}}{R_{max} - R_{min}}$$
$$= \frac{\theta}{T_{max} - T_{min}} \mathbf{M} \mathbf{d} + \frac{1 - \theta}{R_{max} - R_{min}} \mathbf{x}^T \mathbf{W} \mathbf{x}$$
$$+ Const$$
(26)

where  $Const = -\frac{\theta T_{min}}{T_{max} - T_{min}} - \frac{(1-\theta)R_{min}}{R_{max} - R_{min}}$ .  $T_{max}$  and  $R_{max}$  represent the maximum value of image pull delay and communication overhead, respectively.  $T_{min}$  and  $R_{min}$  represent the minimum value of image pull delay and communication overhead.  $\theta \in [0, 1]$  represents the preference for image pull delay and communication overhead. Finally, the original optimization problem can be transformed into a new optimization problem as follows:

P3: 
$$\min_{x,d} F(\mathbf{x}, \mathbf{d})$$
 (27)  
s.t. (9), (19) - (24)

This problem is an integer quadratic programming problem. The solution of P3 is the weakly Pareto optimal solution of the original problem. If  $\theta \in (0, 1)$ , the solution of P2 is the Pareto optimal solution. The proof can be found in [33].

# 4.2 Solution based on successive convex approximation

Since **W** is not a positive semi-definite matrix, the problem is a non-convex quadratic programming, which is difficult to solve directly. First, we transform P3 into a convex optimization problem. Then the problem can be solved based on SCA [34]. Let  $\mathbf{Q} = \mathbf{W} + \mathbf{W}^T$ , then minimizing  $F(\mathbf{x}, \mathbf{d})$ is equivalent to minimize

$$U(\mathbf{x}, \mathbf{d}) = C_1 \mathbf{M} \mathbf{d} + \frac{1}{2} C_2 \mathbf{x}^T \mathbf{Q} \mathbf{x}$$
(28)

where  $C_1 = \frac{\theta}{T_{max} - T_{min}}$ ,  $C_2 = \frac{1 - \theta}{R_{max} - R_{min}}$ . For the matrix **Q**, set the eigenvalues of the matrix as  $\lambda_1, \lambda_2, \dots, \lambda_n$ . We can define  $\lambda_Q = \max\{|\lambda_i|\}$ , and the matrix **Q** can be split as follows:

$$\mathbf{Q} = \mathbf{Q} + \lambda_Q \mathbf{I} - \lambda_Q \mathbf{I} = \mathbf{P} - \mathbf{N}$$
(29)

where  $\mathbf{P} = \mathbf{Q} + \lambda_Q \mathbf{I}$ ,  $\mathbf{N} = \lambda_Q \mathbf{I}$ . Equation (28) becomes

$$U(\mathbf{x}, \mathbf{d}) = U_1(\mathbf{x}, \mathbf{d}) - U_2(\mathbf{x})$$
(30)

where  $U_1(\mathbf{x}, \mathbf{d}) = C_1 \mathbf{M} \mathbf{d} + \frac{1}{2}C_2 \mathbf{x}^T \mathbf{P} \mathbf{x}$  is convex, and  $-U_2(\mathbf{x}) = -\frac{1}{2}C_2 \mathbf{x}^T \mathbf{N} \mathbf{x}$  is nonconvex. Next, we need to make a convex approximation to  $-U_2(\mathbf{x})$  at  $\bar{x}$ , where  $\bar{x}$  is a point in the feasible set of P3. Let

$$l(\mathbf{x}) = -U_2(\bar{\mathbf{x}}) - \nabla U_2(\bar{\mathbf{x}})^T (\mathbf{x} - \bar{\mathbf{x}})$$
  
=  $-C_2 \bar{\mathbf{x}}^T \mathbf{N} \mathbf{x} + \frac{1}{2} C_2 \bar{\mathbf{x}}^T \mathbf{N} \bar{\mathbf{x}} \ge -U_2(\mathbf{x})$  (31)

Finally, we get the convex approximation problem

P4: min 
$$U_{qp}(\mathbf{x}, \mathbf{d}; \bar{\mathbf{x}}, \bar{\mathbf{d}})$$
  

$$= U_1(\mathbf{x}, \mathbf{d}) + l(\mathbf{x})$$

$$= C_1 \mathbf{M} \mathbf{d} + \frac{1}{2} C_2 \mathbf{x}^T \mathbf{P} \mathbf{x}$$

$$- C_2 \bar{\mathbf{x}}^T \mathbf{N} \mathbf{x} + \frac{1}{2} C_2 \bar{\mathbf{x}}^T \mathbf{N} \bar{\mathbf{x}}$$
(32)  
s.t. (9), (19) - (24)

P4 is a convex quadratic programming problem and can be solved directly with the commercial solver. So we can solve P3 by SCA [34] algorithm, as shown in Algorithm 1.

#### 4.3 Convergence analysis

In this subsection, we show that the SCA algorithm can reach the optimal solution of P3.

**Theorem 1.** If  $\bar{\mathbf{x}}$ ,  $\bar{\mathbf{d}}$  is the optimal solution to P4, then  $\bar{\mathbf{x}}$ ,  $\bar{\mathbf{d}}$  is the KKT point of P3.

*Proof.* Proof is provided in Appendix B.  $\Box$ 

**Theorem 2.** The problem P3 can get a stationary solution by Algorithm 1.

*Proof.* Proof is provided in Appendix C.  $\Box$ 

According to Theorem 1 and Theorem 2, we can get that Algorithm 1 can converge to a stationary point and the point is the KKT point of P3. Since the original problem is nonconvex, the global optimal solution cannot be obtained. The solution obtained by Algorithm 1 based on the SCA method [34] is the approximate optimal solution of the original problem.

#### Algorithm 1 Successive convex approximation algorithm

1: Find a feasible point  $\mathbf{x}^0$  and  $\mathbf{d}^0$ , choose a stepsize  $\alpha \in (0, 1]$ , and set  $r = 0, \epsilon > 0$ 

2: repeat  
3: 
$$z_x^{r+1}, z_d^{r+1} = \arg\min_{x} U_{qp}(\mathbf{x}, \mathbf{d}; \mathbf{x}^r, \mathbf{d}^r)$$
  
4:  $x^{r+1} = x^r + \alpha(z_x^{r+1} - x^r)$   
5:  $d^{r+1} = d^r + \alpha(z_x^{r+1} - y^r)$ 

5: 
$$d'^{+1} = d' + \alpha(z'_d^{+1} - y')$$
  
5:  $r \leftarrow r + 1$ 

7: **until** 
$$\|\mathbf{x}^r - \mathbf{x}^{r-1}\| + \|\mathbf{d}^r - \mathbf{d}^{r-1}\| \le \epsilon$$

# 5 **RESOURCE REALLOCATION SCHEME**

We can get a microservice deployment strategy for layer sharing and chain sharing from Sec. 4. However, the computing resources of all servers will not be fully utilized due to the constraints of computing resources and storage resources of edge servers. Edge servers may face the problem that one resource is used up while the other resource is still available. It is a waste of spare resources. In this section, we will propose a server resource redistribution method, which can fully utilize the spare resources of the server.

#### 5.1 Problem formulation

After a microservice is deployed, the deployment location of the microservice remains unchanged until the end of the microservice. Assume that the computing resource of server n is  $C_n^C$ , and J microservices are deployed on it. These microservices can be described by a set  $\mathbb{J} = \{1, \dots, J\}$ . The minimum computing resource requested by microservice j is  $u_j$ , and the computing resource actually allocated to microservice j is  $f_j$ . The computing resources allocated to the microservice must be higher than the computing resources it requests, so there is a constraint  $f_j \ge u_j$ .

Assuming that the amount of data that the microservice needs to process is *Data*. The original processing time required is  $t_{old} = \frac{Data}{u_j}$ , and the new processing time is  $t_{new} = \frac{Data}{f_j}$ . The ratio of the new processing time to the original processing time is  $\frac{t_{new}}{t_{old}} = \frac{u_j}{f_j}$ . So we define the evaluation function  $e_j = \frac{u_j}{f_j}$  to evaluate the impact of allocated computing resources on the processing efficiency of microservices. Then we can define the optimization problem as follows

$$P5:\min U = \sum_{j \in \mathbb{I}} e_j \tag{33}$$

s.t. 
$$f_j \ge u_j, \forall j \in \mathbb{J}$$
 (34)

$$\sum_{j \in \mathbb{I}} f_j \leqslant C_n^C \tag{35}$$

Constraint (35) means that the total computing resources allocated need to be less than the total resources of the server.

#### 5.2 Solution based on Lagrange Multiplier Method

The Lagrangian function of P5 is constructed as

#### Algorithm 2 Greedy deployment strategy

- 1: Normalize the communication data and layer size of each microservice by:  $w_{ij,new}^k = \theta \frac{w_{ij}^k w_{min}}{w_{max} w_{min}}$ ,  $S_{new}^l = (1 \theta) \frac{S^l S_{min}}{2}$
- (1 θ) S<sup>l</sup>-S<sub>min</sub>/S<sub>max</sub> S<sub>min</sub>
   2: Sort w<sup>ki</sup><sub>new</sub> and S<sup>l</sup><sub>new</sub>. The larger the value is, the higher the priority will be. Each value corresponds to two microservices with a large amount of communication data or several groups of microservices with a larger image layer. Get the sorted list *List*
- 3: for ms in List do
- 4: Check if these microservices in *ms* have been deployed
- 5: if all microservices have been deployed then
- 6: continue
- 7: **else**
- 8: **if** microservices in ms are from  $w_{ij,new}^k$  **then**
- 9: Find the closest server *n* where the microservice is deployed and deploy microservice in *n*.
- 10: **if** server *n* has no enough resource **then**
- 11: find a closet server from server n to deploy.
- 12: end if
- 13: else
- 14: Deploy microservice in the server which has maximum  $b_n^{cloud}$  and enough resource
- 15: **end if**
- 16: end if
- 17: end for
- 18: Output deployment strategy

$$L(f_j, \lambda_i, \mu) = \sum_{j \in \mathbb{J}} e_j + \sum_{j \in \mathbb{J}} \lambda_i (u_j - f_j) + \mu(\sum_{j \in \mathbb{J}} f_j - C_n^C)$$
(36)

Its KKT condition is

$$\begin{cases} \nabla L_{f_j}(f_j, \lambda_i, \mu) \\ = -\sum_{j \in \mathbb{J}} \frac{u_j}{f_j^2} + \sum_{j \in \mathbb{J}} \lambda_i f_j + \mu J = 0 \\ \lambda_i(u_j - f_j) = 0, \forall i \in \mathbb{J} \\ \mu(\sum_{j \in \mathbb{J}} f_j - C_n^C) = 0 \\ \lambda_i \ge 0, \forall i \in \mathbb{J} \\ \mu \ge 0 \end{cases}$$
(37)

By solving (37), we can get

$$f_j = \frac{u_j}{\sum_{j \in \mathbb{J}} u_j} C_n^C \tag{38}$$

A simple explanation is that each microservice is proportionally multiplied by the ratio of the total computing resources to the initial request resources. In this way, we can redistribute the computing resources and make full use of the computing resources.

## 6 PERFORMANCE EVALUATION

In this section, we evaluate the performance to verify the effectiveness of our proposed method. We use *Gurobi* [35] to

- Greedy Deployment Strategy [32] (GDS): A deployment strategy based on the greedy strategy in [32]. We modified it to fit the experiments in this paper. This algorithm works by weighting the size of the layer and chain. The steps of the algorithm are shown in Algorithm 2.
- Layer-match Scheduling [36] (LS): For each microservice, select an edge server with most amount of its image layers stored locally and sequence layers according to the assignment order.
- Kubernetes Deployment Strategy [37] (K8S): Kubernetes default deployment policy schedules microservices to edge servers with the required images stored locally, otherwise, to the edge server with the least total download size.
- Layer-sharing Deployment Strategy (LDS): A deployment strategy that only considers layer sharing. It is a special case of the proposed method when *θ* = 1.
- Chain-sharing Deployment Strategy (CDS): A deployment strategy that only considers chain sharing. It is a special case of the proposed method when *θ* = 0.

We use Python to conduct simulation experiments on multiple servers to evaluate the performance of the proposed method under different conditions. To accurately evaluate the effectiveness in the real world, we conduct experiments with five real edge servers. Furthermore, we carried out large-scale simulation tests on 15 servers to evaluate the adaptability of the algorithm in large-scale scenarios.

#### 6.1 Simulation experiment

#### 6.1.1 Experimental environment

The experimental platform is Python 3.9.12. The experiments are carried out on a CentOS 7 system equipped with Inter 4210R, 2.40GHz, and 64 GB RAM. We simulated a smart manufacturing production scenario with up to 9 edge servers and 36 microservices. The average storage resource of edge servers is 8 GB, the average computing resource (CPU frequency) of each server is 1.8 GHz with 4 cores, and the bandwidth between the server and the cloud server is 80-200 MB/s. The hop of adjacent servers is 1, and the element values of the **D** matrix vary from 0 to 5, which means the maximum hop of servers is 5. Each application consists of 2-6 microservices, and the communication data between microservices ranges from 100-2000 KB. The computing resource requirements of each microservice range from 0.002 GHz to 1.0 GHz. The number of layers of each microservice is in the range of 6-13. By dividing these layers into shareable and unshareable layers, each image can be regarded as a microservice composed of 1-2 layers. This can reduce the difficulty of calculation. The size of each layer varies from 1-1220 MB. The above data are randomly generated in each experiment to verify the stability of the proposed method. The specific experimental parameter settings are shown in Table 3.



Fig. 4: Objective function value with different condition



Fig. 5: Total image pull delay and total communication overhead with different  $\theta$ 

Symbol	Value
K	4-9
N	4-9
$A_k$	2-6
$C_n^C$	1.4-2.2 GHz
$C_n^S$	4-16 GB
$b_n^{cloud}$	120-200 MB/s
$S^l$	1-1220 MB
$u^{ki}$	0.002-1.0 GHz

**TABLE 3: Experimental parameter** 

#### 6.1.2 Experimental results

Fig. 4 shows the objective function value of different methods. Fig. 4(a) shows the objective function value in different production scale. We take the minimum amount of microservices  $ms_{min} = 12$  and the amount of servers  $n_{min} = 4$  in this experiment as the benchmark values. Then the scale can be described as  $Scale = \frac{1}{2}(\frac{N_{ms}}{ms_{min}} + \frac{N_n}{n_{min}})$ , where  $N_{ms}$  is the amount of microservices, and  $N_n$  is the amount of servers. As can be seen from the figure, our proposed deployment strategy can minimize the objective function compared with the other five methods. The proposed method can also reach relatively stable results under different numbers of microservices and servers. The GDS method can also achieve a good deployment strategy by weighting the layer size and communication overhead. However, due to its greedy strategy, the optimal solution may not always be obtained. The LDS method and the CDS method cannot make the objective function optimal because they only consider one aspect of resource sharing. The LS and K8S methods only consider layer sharing and can not get a better result due to the high communication overhead.

Fig. 4(b) shows the objective value with different microservices and nine servers. It simulates different production loads. The higher the number of microservices is, the higher the load on one server will be. We can see that the proposed deployment strategy can achieve the optimal objective function value, and the value fluctuates within a small range under different microservice loads, which shows that the proposed method is suitable for different load conditions and has good stability. The results of other methods are worse than the proposed method.

Fig. 4(c) shows the objective value with different  $\theta$  when there are 9 servers and 36 microservices on average. It simulates the effect of different weights for microservice image pull delay and communication overhead. In this figure, the function values of the LDS method, LS method, and K8S method change linearly because they only consider layer sharing of the objective function. And CDS method only considers chain sharing of the objective function. So the changes of  $\theta$  can not impact the deployment strategy.



(a) Figure of edge servers

Fig. 6: Figure and topology of edge servers



(a) Delay with different storage ca- (b) Overhead with different storpacity age capacity

Fig. 7: Delay and overhead with different storage capacity

The proposed method can achieve optimal results no matter what value  $\theta$  takes. When  $\theta = 0$ , there is only the chain sharing part in the objective function, and the objective function value is the same as the CDS method. When  $\theta = 1$ , there is only the layer sharing part in the objective function, and the objective function value is the same as the LDS method.

Fig. 5 shows the image pull delay and communication overhead with different  $\theta$ . Since the LDS, CDS, LS, and K8S methods are unaffected by  $\theta$ , the values of these two strategies do not change a lot in the two figures. The fluctuation of the line is more due to randomly generated microservice data. Fig. 5(a) shows the total image pull delay with different  $\theta$ . The higher the weight  $\theta$ , the lower the image pull delay of both the proposed strategy and the GDS method. When  $\theta = 1$ , the proposed deployment strategy can achieve the same result as the LDS method. The image pull delay can be reduced by 140s compared to the CDS method. The proposed strategy can reduce the total image pull delay by 52s on average compared to the GDS method. Fig. 5(b) shows the total communication overhead with different  $\theta$ . The lower the weight  $\theta$ , the lower the total communication overhead of both the proposed strategy and the GDS method. When  $\theta = 0$ , the proposed deployment strategy can achieve the same result as the CDS method. The total communication overhead can be reduced by 80 MB compared to the LDS method. The proposed strategy can reduce total communication overhead by 10 MB on average compared to the GDS method.



(a) Delay with different computing (b) Overhead with different comcapacity puting capacity





Fig. 9: The impact of resource reallocation strategy on the computing time of microservices

#### 6.2 Experiment with Real Edge Servers

#### 6.2.1 Experimental environment

We further conduct experiments with five edge servers in real world to evaluate the effectiveness of our method. The servers are shown in Fig. 6(a) and the topology of servers is shown in Fig. 6(b). Each server has an i5-8250U CPU, 8G RAM, and is equipped with Docker CE. Communication between servers is carried out using the TCP protocol. We select 23 microservices from Docker Hub. The image size of these microservices is in the range of 1.24-1098 MB, and the computation resource requirement is in the range of 0.2-1.4 GHz. The number of layers of each microservice is in the range of 4-11. We simulate servers with different storage and resource constraints by limiting the resource usage of docker. The experimental results are shown in the following figures. Each data point in the figures is the average of multiple experiments.

#### 6.2.2 Experimental results

Fig. 7 shows the total image pull delay and communication overhead under different storage capacities with  $\theta = 0.5$ . As can be seen from Fig. 7(a), the results of the proposed method and the LDS method are very close. When storage capacity becomes more and more sufficient, the proposed method can significantly reduce the image pull delay compared with other methods. This is because when the storage resources are sufficient, more microservices with the same layer can be deployed on one edge server. It can reduce the size of the image layer to be pulled and reduce the delay. In Fig. 7(b), the CDS method can achieve the lowest communication overhead because it is optimized for the

Microservice number (#1)	computing resources (GHz)	time (s)	resources after reallocation (GHz)	time (s)	Microservice number (#2)	computing resources (GHz)	time (s)	resources after reallocation (GHz)	time (s)
1	0.4	16.84	0.576	12.715	1	0.4	17.203	0.554	14.603
2	0.6	35.6	0.864	18.02	2	1	14.687	1.385	14.563
3	1.2	7	1.728	8.572	3	0.6	16.791	0.83	14.516
4	0.8	15.26	1.152	12.567	4	0.8	16.558	1.1	14.508
5	0.4	41.4	0.576	31.934	5	1.2	14.537	1.66	15.156
6	0.7	11.31	1.008	11.171	6	0.6	17.443	0.83	14.566
7	0.9	9.26	1.296	9.613	7	0.6	14.617	0.83	14.544
total	5	136.67	7.2	104.592	total	5.2	111.836	7.2	102.456



Fig. 10: Figure and topology of edge servers

microservice chain. Compared with other methods except the CDS method, the proposed method can achieve lower communication overhead.

Fig. 8 shows the image pull delay and communication overhead under different computing capacities with  $\theta = 0.5$ . The trend is the same as that in Fig. 7. From Fig. 7 and Fig. 8, we can also find that the LDS method has the best effect on image pull delay and the worst effect on communication overhead, and the CDS method is the opposite. This is the disadvantage of not considering both aspects comprehensively. Different from other methods, our proposed method can always obtain a better solution that can better balance the image pull delay and communication overhead.

Fig. 9 shows the impact of the resource reallocation strategy on the completion time of tasks. We selected two from five servers and showed the effect of resource reallocation on their task computing time. The data are shown in Table 4. We can see that resource reallocation can significantly reduce the completion time of computing tasks. In Table 4, the total task completion time of the two servers is reduced from 136.67 seconds and 111.836 seconds to 104.592 seconds and 102.456 seconds, which is a reduction of 23.7% and 8.4%, respectively. This shows that the resource reallocation strategy can effectively reduce the completion time of computing tasks and improve computing efficiency.

#### 6.3 Large-scale Cases

To evaluate the performance of our proposed method in a larger scale scenario [32], we consider the topology of US NSFNET consisting of 15 edge servers as shown in Fig. 10(a). The storage resources of each edge server are 16 GB, the computing resources are 4-core 1.6 GHz, and the bandwidth is 80-120 MB/s. We considered up to 105 microservices selected from Docker Hub and randomly combined them into applications. Other parameter settings are the same as in the previous experiments.

#### 6.3.1 Experimental results

To verify the stability of the proposed method, we conducted ten experiments, and the data for each experiment are presented in Fig. 10. Fig. 10(b) shows the objective function value for ten experiments, it can be seen that the proposed method has little fluctuation and can achieve the lowest objective function value. Since the microservice composition of each experiment is random, the results vary drastically in Fig. 10(c) and Fig. 10(d). The proposed method can achieve almost the same results as the LDS method in terms of image pull delay. It can also achieve similar results to the CDS method in terms of total communication overhead. The results of the proposed method are also better than other schemes.

Fig. 11 shows the ratio of image pull delay and communication overhead to the baseline under different servers and different  $\theta$ . The baseline of image pull delay is defined as the total data size in the absence of layer sharing divided by the average bandwidth of servers. The lower the ratio is, the higher the layer sharing rate will be. As can be seen from the figure, our proposed method can significantly reduce the image pull delay. The image pull delay in the best case is only 56% of the baseline. When  $\theta = 0.5$ , an average of 65% of the baseline ratio can be achieved in the proposed method. The baseline of communication overhead is defined as the summation of all microservice commu-



Fig. 11: The ratio of image pull delay and communication overhead to baseline in different condition

nication data. This ratio can reflect the average number of hops between microservices. The lower the ratio is, the higher the chain sharing rate will be. As can be seen from the figure, our proposed method can significantly reduce communication overhead. The communication overhead in the best case is only 5% of the baseline. When  $\theta = 0.5$ , an average of 30% of the baseline value ratio can be achieved in the proposed method. Moreover, The image pull delay and communication overhead can be reduced significantly in any server numbers, which proves the stability of our proposed method.

Fig. 12 shows the space and time consumption of the proposed method and GDS method to get the deployment strategy under different servers. With the increase in the number of servers and microservices in the network, the amount of layers and the communication between microservices is also increasing. Then the network structure becomes more and more complex. When the amount of servers is less than 10, the computing time of the proposed method is less than 20 seconds and the space occupation is less than 100 MB, which has excellent solution efficiency. In a large-scale server network, the solution time will slow down to about 450 seconds, and the space occupation will also increase to 1800 MB due to the complexity of the network. The optimal solution can still be solved in an acceptable time because the scheduling strategy of microservices does not change frequently in large-scale production. However, since the proposed method is based on solving quadratic programming problems, the time and space complexity is higher than that of Algorithm 2. It is our future work to further optimize the time complexity and space complexity of the proposed method.

# 7 DISCUSSION

In this section, we will discuss the limitations of the proposed method and the future work.

Compared with related works, our proposed method is able to optimize the communication overhead while optimizing the image pull delay. The experiments in Section. 6 also show that the proposed method can achieve the lowest objective function value with a good trade-off between delay and overhead. However, the time and space complexity



Fig. 12: Space and time consumption under different amount of servers

of this method is high in large-scale scenarios. Although the optimal solution can be obtained within an acceptable time, further optimization is required in the future.

The communication overhead depends not only on the amount of communication data, but also on the request frequency. If the request frequency is high, the communication overhead will be very large even if the amount of communication data at one time is small. Therefore, we believe that the calculation of communication overhead should consider the request frequency, that is, to consider the product of the communication data volume and the request frequency. However, since our method is based on microservices not requests, we cannot directly get the request frequency. Therefore, we believe that future work can consider the request frequency as an input, and then use the product of the request frequency and the amount of communication data as the calculation of communication overhead.

Deployed microservices are not static, and the entire production process will include shutdown, migration, and startup of new microservices. In the face of the dynamic microservice deployment process, it is necessary to have corresponding deployment algorithms to adapt to dynamic scenarios. One possible future research direction is to use artificial intelligence or other methods for training after the initial deployment results such that the microservices can be dynamically adjusted.

#### 8 CONCLUSION

In this paper, we study the layer sharing and chain sharing of microservices and explore a microservice deployment scheme that can balance the two ways of resource sharing. We build an image pull delay and communication overhead minimization problem. We transform the problem into a linearly constrained integer quadratic programming problem through model reconstruction and obtain the deployment strategy through a successive convex approximation (SCA) method. Further, we propose a resource reallocation algorithm to fully utilize the idle resources of the server. Experimental results show that the proposed deployment strategy can balance the two resource sharing methods of microservices. When considering the two sharing methods in balance, the average image pull delay can be reduced to 65% of the baseline, and the average communication overhead can be reduced to 30% of the baseline. In the future, we will expand microservice deployment from static scenarios to highly dynamic scenarios and try to obtain rapid solution algorithms in large-scale scenarios.

## ACKNOWLEDGMENT

This work was supported by the National Key Research and Development Program of China (Grant No.2018YFB1702300), and in part by the NSF of China (Grants No. 61731012, 62025305, 61933009, and 92167205).

# APPENDIX A MATRIX VALUE

1)

$$Q = \begin{bmatrix} \mathbf{Q}^1 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \mathbf{Q}^K \end{bmatrix}_{K \times K}$$
(39)

where

$$\mathbf{Q}^{\mathbf{k}} = \begin{bmatrix} \mathbf{q} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \mathbf{q} \end{bmatrix}_{A_k \times A_k}$$
$$\mathbf{q} = [1, 1, \cdots, 1]_{1 \times N}$$

 $\mathbf{b}_1 = [1, 1, \cdots, 1]_{1 \times \sum_{k \in \mathbb{K}} A_k}^T$ 

2)

3)

 $\mathbf{H} = \begin{bmatrix} \mathbf{H}^1 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \mathbf{H}^K \end{bmatrix}_{K \times K}$ (4)

where

 $\mathbf{H}^k = [\mathbf{I}_{n \times n} \quad \mathbf{0} \quad \cdots \quad \mathbf{0}]_{1 \times A_k}$ 

4)

$$\mathbf{b}_2 = [\mathbf{x}^{1,0},\cdots,\mathbf{x}^{K0}]^T \tag{42}$$

where  $\mathbf{x}^{k0} = [0, \cdots, 1, \cdots, 0]^T$ ,  $x_{N^k}^{k0} = 1$ 5)

$$\mathbf{Y} = [\mathbf{Y}_1, \cdots, \mathbf{Y}_N]^T \tag{43}$$

where

$$\begin{split} \mathbf{Y}_{n} &= \begin{bmatrix} \mathbf{P}_{n}^{1} \mathbf{V}_{1}^{1} \mathbf{E}^{1} & \cdots & \mathbf{P}_{n}^{1} \mathbf{V}_{L}^{1} \mathbf{E}^{1} \\ \vdots & \ddots & \vdots \\ \mathbf{P}_{n}^{K} \mathbf{V}_{1}^{K} \mathbf{E}^{K} & \cdots & \mathbf{P}_{n}^{K} \mathbf{V}_{L}^{K} \mathbf{E}^{K} \end{bmatrix}_{K \times L} \\ \mathbf{P}_{n}^{k} &= \begin{bmatrix} \mathbf{p}_{n}^{N} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbf{p}_{n}^{N} \end{bmatrix}_{A_{k} \times A_{k}} \\ \mathbf{p}_{n}^{N} &= [0, \cdots, 0, 1, 0, \cdots, 0]_{1 \times N}^{T}, \mathbf{p}_{n}^{N}(n) = 1 \\ \mathbf{q} &= [1, 1, \cdots, 1]_{1 \times N} \\ \mathbf{V}_{l}^{k} &= \begin{bmatrix} (\mathbf{p}_{l}^{L})^{T} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & (\mathbf{p}_{l}^{L})^{T} \end{bmatrix}_{A_{k} \times A_{k}} \\ \mathbf{p}_{l}^{L} &= [0, \cdots, 0, 1, 0, \cdots, 0]_{1 \times L}^{T}, \mathbf{p}_{l}^{L}(l) = 1 \\ \mathbf{E}^{k} &= \begin{bmatrix} (\mathbf{E}^{k1})^{T}, \cdots, (\mathbf{E}^{kA_{k}})^{T} \end{bmatrix}^{T} \\ \mathbf{E}^{ki} &= \begin{bmatrix} E^{ki1}, \cdots, E^{kiL} \end{bmatrix}^{T} \end{split}$$

6)

$$S = \begin{bmatrix} \mathbf{S}^T & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \mathbf{S}^T \end{bmatrix}_{N \times N}$$
(44)

7)

where

$$\mathbf{C}^{S} = \left[C_{1}^{S}, \cdots, C_{N}^{S}\right]^{T}$$
(45)

(46)

(47)

8)

where

$$\mathbf{G}_{n} = \left[ \left( \mathbf{P}_{n}^{1} \mathbf{u}^{1} \right)^{T}, \cdots, \left( \mathbf{P}_{n}^{K} \mathbf{u}^{K} \right)^{T} \right]^{T}$$
$$\mathbf{u}^{k} = \left[ u^{k1}, \cdots, u^{kA_{k}} \right]^{T}$$
$$\mathbf{C}^{C} = \left[ C_{1}^{C}, \cdots, C_{N}^{C} \right]^{T}$$

 $\mathcal{G} = [\mathbf{G}_1, \cdots, \mathbf{G}_N]^T$ 

 $\mathbf{S} = \begin{bmatrix} S^1, \cdots, S^L \end{bmatrix}^T$ 

9)

(40)

# (41) APPENDIX B PROOF OF THEOREM 1

*Proof.* We remove the bolding of all letters in this proof to simplify the expression. We combine the same parts of the constraints in P3 and rewrite it as

$$\min F(x,d) = C_1 M d + \frac{1}{2} C_2 x^T P x$$
$$- C_2 \bar{x}^T N x + \frac{1}{2} C_2 \bar{x}^T N \bar{x}$$
s.t.  $a_i^T x = b_i, i = 1, \cdots, m$ 
$$a_i^T x \leqslant b_i, i = m, \cdots, m + l$$
$$c_i^T y \leqslant g_i, i = 1, \cdots, n$$
$$e_i x + f_i y \leqslant 0, i = 1, \cdots, p$$

Its KKT condition is

$$\begin{cases} C_2 P \bar{x} - C_2 N \bar{x} + \sum_{i=1}^{m+l} \lambda_i a_i + \sum_{i=1}^k \nu e_i = 0 \\ C_1 M + \sum_{i=1}^n \mu_i c_i + \sum_{i=1}^k \nu e_i = 0 \\ a_i^T x = b_i, i = 1, \cdots, m \\ a_i^T x \leqslant b_i, i = m, \cdots, m+l \\ c_i^T y \leqslant g_i, i = 1, \cdots, n \\ e_i x + f_i y \leqslant 0, i = 1, \cdots, p \\ \lambda_i \geqslant 0, i = 1, \cdots, m+l \\ \mu_i \geqslant 0, i = 1, \cdots, n \\ \nu_i \geqslant 0, i = 1, \cdots, p \\ \lambda_i (a_i^T x - b_i) = 0, i = m, \cdots, m+l \\ \mu_i (c_i^T y - g_i) = 0, i = 1, \cdots, n \\ \nu_i (e_i x + f_i y) = 0, i = 1, \cdots, p \end{cases}$$
(48)

Since  $P\bar{x} - N\bar{x} = Q\bar{x}$ , so

$$\begin{cases} C_2 Q \bar{x} + \sum_{i=1}^{m+l} \lambda_i a_i + \sum_{i=1}^k \nu e_i = 0\\ C_1 M + \sum_{i=1}^n \mu_i c_i + \sum_{i=1}^k \nu e_i = 0\\ a_i^T x = b_i, i = 1, \cdots, m\\ a_i^T x \leqslant b_i, i = m, \cdots, m + l\\ c_i^T y \leqslant g_i, i = 1, \cdots, n\\ e_i x + f_i y \leqslant 0, i = 1, \cdots, p\\ \lambda_i \ge 0, i = 1, \cdots, m + l\\ \mu_i \ge 0, i = 1, \cdots, n\\ \nu_i \ge 0, i = 1, \cdots, p\\ \lambda_i (a_i^T x - b_i) = 0, i = m, \cdots, m + l\\ \mu_i (c_i^T y - g_i) = 0, i = 1, \cdots, n\\ \nu_i (e_i x + f_i y) = 0, i = 1, \cdots, p \end{cases}$$
(49)

Therefore,  $\bar{x}$  is the KKT point of P3, the non-global solution of the original problem can be obtained.

# APPENDIX C PROOF OF THEOREM 2

*Proof.* We remove the bolding of all letters in this proof to simplify the expression. We can find that the search direction of the algorithm is  $\lambda_x^{r+1} = z_x^{r+1} - x^r$ ,  $\lambda_y^{r+1} = z_d^{r+1} - d^r$ . According to the convexity of P4, we can get

$$U(x^{r+1}, d^{r+1}) \leq U_{qp}(x^{r+1}, d^{r+1}; x^{r}, d^{r}) \leq \alpha U_{qp}(x^{r+1}, z_{d}^{r+1}; x^{r}, d^{r}) + (1 - \alpha) U_{qp}(x^{r+1}, d^{r}; x^{r}, d^{r}) \leq U_{qp}(x^{r+1}, d^{r}; x^{r}, d^{r}) \leq \alpha U_{qp}(z_{x}^{r+1}, d^{r}; x^{r}, d^{r}) + (1 - \alpha) U_{qp}(x^{r}, d^{r}; x^{r}, d^{r}) \leq U_{qp}(x^{r}, d^{r}; x^{r}, d^{r}) = U(x^{r}, d^{r})$$
(50)

where the second and the fourth inequality sign come from the convexity of  $U_{qp}(\cdot; x^r, y^r)$ . Then the value of the objective function must not be monotone increasing.

$$U(x^{r}, d^{r}) - U(x^{r+1}, d^{r+1}) = U(x^{r}, d^{r}) - U(x^{r+1}, d^{r}) + U(x^{r+1}, d^{r}) - U(x^{r+1}, d^{r+1}) = U(x^{r}, d^{r}) - U(x^{r} + \alpha(z_{x}^{r+1} - x^{r}), d^{r}) + U(x^{r+1}, d^{r}) - U(x^{r+1}, d^{r} + \alpha(z_{d}^{r+1} - d^{r})) = -\alpha U'(x^{r}, d^{r}; \lambda_{x}^{r+1}) - \alpha U'(x^{r+1}, d^{r}; \lambda_{d}^{r+1}) = 0$$

$$(51)$$

Then we can get

$$U(x^*, d^*) - U(x^0, d^0) \\ \leqslant U(x^r, d^r) - U(x^0, d^0) \\ \leqslant \sum_{r=0}^r \alpha(U'(x^r, d^r; \lambda_x^{r+1}) + U'(x^{r+1}, d^r; \lambda_d^{r+1}))$$
(52)

thereby

$$\lim_{r \to \infty} U'(x^r, d^r; \lambda_x^{r+1}) = \lim_{k \to \infty} U'(x^r, d^r; \lambda_y^{r+1}) = 0 \quad (53)$$

So problem P3 can get a stationary solution by SCA algorithm. We can choose  $\alpha = 1$  for integer variables, and it still holds.

## REFERENCES

- R. Wang, L. Ji, T. Ren, S. He, and Z. Shi, "A Low-latency and Interoperable Industrial Internet of Things Architecture for Manufacturing Systems," in 2020 IEEE 18th International Conference on Industrial Informatics (INDIN). Warwick, United Kingdom: IEEE, Jul. 2020, pp. 859–864.
- [2] L. Chen, Z. Lu, A. Xiao, Q. Duan, J. Wu, and P. C. K. Hung, "A Resource Recommendation Model for Heterogeneous Workloads in Fog-Based Smart Factory Environment," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 3, pp. 1731–1743, Jul. 2022.
- [3] N. C. Mendonca, P. Jamshidi, D. Garlan, and C. Pahl, "Developing Self-Adaptive Microservice Systems: Challenges and Directions," *IEEE Software*, vol. 38, no. 2, pp. 70–79, Mar. 2021.
- [4] Z. Nie and K.-C. Chen, "Hypergraphical Real-time Multi-Robot Task Allocation in a Smart Factory," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2021.
- [5] A. Hazra, P. K. Donta, T. Amgoth, and S. Dustdar, "Cooperative Transmission Scheduling and Computation Offloading with Collaboration of Fog and Cloud for Industrial IoT Applications," *IEEE Internet of Things Journal*, pp. 1–1, 2022.
  [6] K. Thramboulidis, D. C. Vachtsevanou, and A. Solanos, "Cyber-
- [6] K. Thramboulidis, D. C. Vachtsevanou, and A. Solanos, "Cyberphysical microservices: An IoT-based framework for manufacturing systems," in 2018 IEEE Industrial Cyber-Physical Systems (ICPS), 2018, pp. 232–239.
- [7] Z. Yang, P. Nguyen, H. Jin, and K. Nahrstedt, "MIRAS: Modelbased Reinforcement Learning for Microservice Resource Allocation over Scientific Workflows," in 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), Jul. 2019, pp. 122–132.
- [8] H. Tian, X. Xu, T. Lin, Y. Cheng, C. Qian, L. Ren, and M. Bilal, "DIMA: Distributed cooperative microservice caching for internet of things in edge computing by deep reinforcement learning," *World Wide Web*, Aug. 2021.
- [9] L. Bao, C. Wu, X. Bu, N. Ren, and M. Shen, "Performance Modeling and Workflow Scheduling of Microservice-Based Applications in Clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 9, pp. 2114–2129, Sep. 2019.
- [10] C. Jian, J. Ping, and M. Zhang, "A cloud edge-based two-level hybrid scheduling learning model in cloud manufacturing," *International Journal of Production Research*, vol. 59, no. 16, pp. 4836–4850, Aug. 2021.
- [11] W. Šhi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

- [12] C. Zhang, X. Liu, X. Zheng, R. Li, and H. Liu, "FengHuoLun: A Federated Learning based Edge Computing Platform for Cyber-Physical Systems," in 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Mar. 2020, pp. 1–4.
- [13] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," Linux Journal, vol. 2014, no. 239, p. 2:2, Mar. 2014.
- [14] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," Communications of the ACM, vol. 59, no. 5, pp. 50-57, Apr. 2016.
- [15] L. Gu, D. Zeng, J. Hu, H. Jin, S. Guo, and A. Y. Zomaya, "Exploring Layered Container Structure for Cost Efficient Microservice Deployment," in IEEE INFOCOM 2021 - IEEE Conference on Computer Vancouver, BC, Canada: IEEE, May 2021, pp. Communications. 1 - 9
- [16] L. Gu, Q. Tang, S. Wu, H. Jin, Y. Zhang, G. Shi, T. Lin, and J. Rao, "N-Docker: A NVM-HDD Hybrid Docker Storage Framework to Improve Docker Performance," in Network and Parallel Computing, ser. Lecture Notes in Computer Science, X. Tang, Q. Chen, P. Bose, W. Zheng, and J.-L. Gaudiot, Eds. Cham: Springer International
- Publishing, 2019, pp. 182–194. [17] Y. Wang, C. Zhao, S. Yang, X. Ren, L. Wang, P. Zhao, and X. Yang, "MPCSM: Microservice Placement for Edge-Cloud Collaborative Smart Manufacturing," IEEE Transactions on Industrial Informatics, vol. 17, no. 9, pp. 5898–5908, Sep. 2021.
- [18] W. Lv, Q. Wang, P. Yang, Y. Ding, B. Yi, Z. Wang, and C. Lin, "Microservice Deployment in Edge Computing Based on Deep Q Learning," IEEE Transactions on Parallel and Distributed Systems, pp. 1-1, 2022
- [19] Y. Yu, J. Yang, C. Guo, H. Zheng, and J. He, "Joint optimization of service request routing and instance placement in the microservice system," Journal of Network and Computer Applications, vol. 147, p. 102441, Dec. 2019.
- [20] J. L. Herrera, J. Galán-Jiménez, J. Berrocal, and J. M. Murillo, "Optimizing the Response Time in SDN-Fog Environments for Time-Strict IoT Applications," IEEE Internet of Things Journal, vol. 8, no. 23, pp. 17172–17185, Dec. 2021.
- [21] S. Deng, Z. Xiang, J. Taheri, M. A. Khoshkholghi, J. Yin, A. Y. Zomaya, and S. Dustdar, "Optimal Application Deployment in Resource Constrained Distributed Edges," IEEE Transactions on Mobile Computing, vol. 20, no. 5, pp. 1907–1923, May 2021.
- [22] X. Chen, Y. Bi, X. Chen, H. Zhao, N. Cheng, F. Li, and W. Cheng, "Dynamic Service Migration and Request Routing for Microservice in Multi-cell Mobile Edge Computing," IEEE Internet of Things Journal, pp. 1-1, 2022.
- [23] E. Fadda, P. Plebani, and M. Vitali, "Monitoring-Aware Optimal Deployment for Applications Based on Microservices," IEEE Transactions on Services Computing, vol. 14, no. 6, pp. 1849–1863, 2021.
- [24] P. Zhao, P. Wang, X. Yang, and J. Lin, "Towards Cost-Efficient Edge Intelligent Computing With Elastic Deployment of Container-Based Microservices," IEEE Access, vol. 8, pp. 102947–102957, 2020.
- [25] F. Faticanti, F. De Pellegrini, D. Siracusa, D. Santoro, and S. Cretti, "Throughput-Aware Partitioning and Placement of Applications in Fog Computing," IEEE Transactions on Network and Service Management, vol. 17, no. 4, pp. 2436–2450, Dec. 2020.
- [26] C. T. Joseph and K. Chandrasekaran, "IntMA: Dynamic Interaction-aware resource allocation for containerized microservices in cloud environments," Journal of Systems Architecture, vol. 111, p. 101785, Dec. 2020.
- [27] X. Li, Z. Zhou, C. Zhu, L. Shu, and J. Zhou, "Online Reconfiguration of Latency-Aware IoT Services in Edge Networks," IEEE Internet of Things Journal, pp. 1–12, 2021.
- [28] V. Armani, F. Faticanti, S. Cretti, S. Kum, and D. Siracusa, "A Cost-Effective Workload Allocation Strategy for Cloud-Native Edge Services," arXiv:2110.12788 [cs], Oct. 2021.
- [29] M. Sasabe and T. Hara, "Capacitated Shortest Path Tour Problem-Based Integer Linear Programming for Service Chaining and Function Placement in NFV Networks," IEEE Transactions on Network and Service Management, vol. 18, no. 1, pp. 104–117, Mar. 2021.
- [30] J. Lou, H. Luo, Z. Tang, W. Jia, and W. Zhao, "Efficient Container Assignment and Layer Sequencing in Edge Computing," IEEE *Transactions on Services Computing*, pp. 1–1, 2022. [31] L. Gu, D. Zeng, J. Hu, B. Li, and H. Jin, "Layer Aware Microser-
- vice Placement and Request Scheduling at the Edge," in IEEE

- [32] L. Gu, Z. Chen, H. Xu, D. Zeng, B. Li, and H. Jin, "Layeraware Collaborative Microservice Deployment toward Maximal Edge Throughput," in IEEE INFOCOM 2022 - IEEE Conference on Computer Communications, 2022, pp. 71-79.
- [33] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-Aware Microservice Coordination in Mobile Edge Computing: A Reinforcement Learning Approach," *IEEE Transactions on Mobile Computing*, vol. 20, no. 3, pp. 939–951, Mar. 2021.
- [34] M. Razaviyayn, "Successive convex approximation: Analysis and applications," University of Minnesota, May 2014.
- "Gurobi The Fastest Solver," https://www.gurobi.com/.
- [36] S. Fu, R. Mittal, L. Zhang, and S. Ratnasamy, "Fast and Efficient Container Startup at the Edge via Dependency Scheduling," 3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20), p. 7, 2020.
- [37] "Kubernetes, Kubernetes." https://kubernetes.io/.



Yuxiang Liu received the B.Eng. degree in control science and engineering from Shanghai Jiao Tong University, Shanghao, China, in 2020.

He is currently pursuing the Ph.D. degree at the Department of Automation, Shanghai Jiao Tong University, Shanghai, China. His current research interests include microservice deployment, intelligent manufacturing.



Bo Yang (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the City University of Hong Kong, Hong Kong, in 2009.

He is currently a Full Professor with Shanghai Jiao Tong University, Shanghai, China. Prior to joining Shanghai Jiao Tong University in 2010, he was a Post-Doctoral Researcher with the KTH Royal Institute of Technology, Stockholm, Sweden, from 2009 to 2010, and a Visiting Scholar with the Polytechnic Institute of New

York University in 2007. His research interests include game theoretical analysis and optimization of energy networks and wireless networks. He is on the Editorial Board of Digital Signal Processing and in TPC of several international conferences. He has been the Principle Investigator in several research projects, including the NSFC Key Project. He was a recipient of the Ministry of Education Natural Science Award 2016, the Shanghai Technological Invention Award 2017, the Shanghai Rising-Star Program 2015, and the SMC-Excellent Young Faculty Award by Shanghai Jiao Tong University.



Yu Wu received the B.Eng. degree at the Department of Automation, Harbin Engineering University, Harbin, China, in 2019. He is currently pursuing the Ph.D. degree at the Department of Automation, Shanghai Jiao Tong University, Shanghai, China. His current research interests include edge computing, industrial Internet of Things, and machine learning for wireless networks.



**Cailian Chen** Cailian Chen (S'03-M'06) received the B. Eng. and M. Eng. degrees in Automatic Control from Yanshan University, P. R. China in 2000 and 2002, respectively, and the Ph.D. degree in Control and Systems from City University of Hong Kong, Hong Kong SAR in 2006. She has been with the Department of Automation, Shanghai Jiao Tong University since 2008. She is now a Distinguished Professor.

Prof. Chen's research interests include industrial wireless networks and computational intel-

ligence, and Internet of Vehicles. She has authored 3 research monographs and over 100 referred international journal papers. She is the inventor of more than 30 patents. Dr. Chen received the prestigious "IEEE Transactions on Fuzzy Systems Outstanding Paper Award" in 2008, and 5 conference best paper awards. She won the Second Prize of National Natural Science Award from the State Council of China in 2018, First Prize of Natural Science Award from The Ministry of Education of China in 2006 and 2016, respectively, and First Prize of Technological Invention of Shanghai Municipal, China in 2017. She was honored "National Outstanding Young Researcher" by NSF of China in 2020 and "Changjiang Young Scholar" in 2015.



Xinping Guan (Fellow, IEEE) received the B.Sc. degree in mathematics from Harbin Normal University, Harbin, China, in 1986, and the Ph.D. degree in control science and engineering from Harbin Institute of Technology, Harbin, China, in 1999.

He is currently the Chair Professor of Shanghai Jiao Tong University, Shanghai, China, where he is the Dean of School of Electronic, Information and Electrical Engineering and the Director of the Key Laboratory of Systems Con-

trol and Information Processing, Ministry of Education of China. Before that, he was the Executive Director of Office of Research Management, Shanghai Jiao Tong University, a Full Professor, and Dean of Electrical Engineering, Yanshan University, Qinhuangdao, China. As a Principal Investigator, he has finished/been working on more than 20 national key projects. He is the leader of the prestigious Innovative Research Team of the National Natural Science Foundation of China. He is an Executive Committee Member of Chinese Automation Association Council and the Chinese Artificial Intelligence Association Council. He has authored or coauthored five research monographs, more than 200 papers in IEEE transactions and other peer-reviewed journals, and numerous conference papers. His current research interests include industrial network systems, smart manufacturing, and underwater networks.

Dr. Guan received the Second Prize of the National Natural Science Award of China in both 2008 and 2018, the First Prize of Natural Science Award from the Ministry of Education of China in both 2006 and 2016. He was a recipient of the IEEE Transactions on Fuzzy Systems Outstanding Paper Award in 2008. He is a National Outstanding Youth honored by NSF of China, and Changjiang Scholar's by the Ministry of Education of China and State-Level Scholar of New Century Bai Qianwan Talent Program of China.