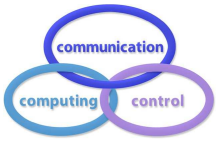
INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
Online ISSN 1841-9844, ISSN-L 1841-9836, Volume: 21, Issue: 5, Month: October, Year: 2026
Article Number: 7222, https://doi.org/10.15837/ijccc.2026.5.7222



CCC Publications

Multi-Objective DRL for Efficient Kubernetes Microservice Scheduling

H. Feng, Y.M. Shi, M.K. Zhen

Hao Feng
School of Computer Science and Technology
Tongji University
4800 Cao'an Highway
Shanghai, China
fenghaozz@126.com

Yunmei Shi*
College of Electronics and Information Engineering
Tongji University
4800 Cao'an Highway
Shanghai, China
ymshi@tongji.edu.cn
*Corresponding author: ymshi@tongji.edu.cn

Mingkai Zhen
School of Computer Science and Technology
Tongji University
4800 Cao'an Highway
Shanghai, China
kfzn@tongji.edu.cn

Abstract

Container based microservice architecture is becoming the priority of service development in edge computing. Its flexibility and scalability could provide stable service response. However, due to the limited computing resources of each edge node in edge computing, how to use computing resources efficiently has become an important issue in microservice deployment. This study formulates the multi-objective microservice deployment problem (MMDP) in Kubernetes, balancing service latency and resource utilization. We propose a deep reinforcement learning method with reward shaping and heuristic scaling to derive optimal deployment policies. Experiments show that our approach reduces response time by 25–30% and improves load balancing by 20% compared with Kubernetes default scheduling and DQL. These results demonstrate the applicability of reinforcement learning for distributed computing and automatic control in microservice systems.

Keywords: microservice, multi-objective microservice deployment model, deep reinforcement learning, elastic scaling.

1 Introduction

Against the backdrop of rapid development in cloud computing, traditional service development methods are gradually being replaced by microservice development methods due to their poor scalability and difficulty in maintenance [10]. The microservice architecture splits a single application into multiple independent microservices, which are called through interfaces and do not affect each other's running results. This method not only improves the flexibility and maintainability of the system, but also reduces the resource redundancy caused by individual applications occupying resources [22, 27]. At the same time, with the rapid development of virtualization technology and edge computing, container based microservice development is becoming the mainstream of microservice design. Containers on different edge nodes could provide virtualized isolation space for individual microservices, ensuring that they do not interfere with each other [24]. Also, each of them has sufficient computing resources [28].

However, the computing resources required for each microservice are irregular, and the total amount of resources for nodes is limited. So how to efficiently utilize computing resources and allocate them reasonably within limited resources has become an important issue in microservice deployment [45]. Traditional microservice deployment mostly involves coarse-grained resource allocation, which can easily lead to resource waste. In addition, while allocating resources reasonably, microservice deployment should also consider quality of service (QOS) [1]. Especially when multiple user requests occur simultaneously, it is necessary to ensure a sufficiently fast service response [37]. Therefore, the efficient microservice deployment method should ensure the faster user request response and the delicate resource utilization.

In recent years, there have been many research focused on microservice deployment. Papers such as [2, 33, 39], take many efforts to reduce service response time effectively. Where the response time includes the processing time and the data transmission time of each service. But there remain some problems. For example, in papers [19, 38, 48], they only considered optimizing resource load balancing and service response latency, the elastic scaling and user access under high concurrency are ignored. In papers [32, 35, 36], they implemented microservice traffic routing under high concurrency and high traffic but without microservice deployment. Kubernetes is a comprehensive tool for microservice orchestration and deployment. Its function is realized on the basis of containers. Each container establishes a microservice, and the containers interact with each other through network communication. It realizes the distributed deployment of microservices and makes full use of the resource advantages of edge computing. Therefore, many studies have optimized and validated microservice deployment on Kubernetes. Such as in papers [7, 23, 31, 42], they tended to employ a microservice deployment followed by a request routing scheduling. Although considering the convenience of deploying Kubernetes services, implementing a microservice deployment method that balances resource balance and service responsiveness in Kubernetes is also a challenge. Therefore, we propose to combine user requests and resource balancing to form a multi-objective optimization problem, in order to design a better microservice deployment strategy.

In order to implement the above microservice deployment strategy, we propose a deep reinforcement learning method based on reward shaping. But there are still some challenges that need to be addressed. Firstly, during periods of large-scale user requests, microservices need to be deployed in real time based on user requests, and they should be distributed in the cluster to ensure service response. Secondly, establishing the multi-objective optimization problem of optimal resource balance and fastest service response requires real-time allocation of all dynamic computing resources of nodes. Finally, when using deep reinforcement learning to address the aforementioned issues, it is necessary to design appropriate reward parameters to ensure real-time performance and optimality.

To solve the above challenge, this paper proposes a deep reinforcement learning method based on reward shaping to solve multi-objective optimization problems, obtain optimal deployment strategies, and design heuristic methods to achieve automatic horizontal elastic scaling of containers. Unlike existing DRL methods that target only latency or load balance, our method simultaneously optimizes both objectives while enabling elastic scaling in high-concurrency scenarios. In summary, the method presented in this article mainly consists of the following three parts.

First, this paper constructs a multi-objective microservice deployment problem (MMDP) for stage

resource balancing and rapid service response. This problem establishes the goal of fast service response based on the service response delay obtained after processing the Jackson network queue; And resource balancing is based on balancing all computing resources of the nodes, including CPU, memory, IO and network transmission rate. Moreover, it allows the deployment number of individual microservices to scale elastically according to the dynamic changes in high concurrency state.

Second, this paper proposes a deep reinforcement learning algorithm based on reward shaping to solve the deployment problem of multi-objective microservices under high concurrency scenarios. This algorithm applies Markov theory to construct a multi-objective service deployment problem as a Markov decision process with complex state, action, and reward structures, and then uses deep learning for global solution to obtain the optimal microservice deployment strategy.

Third, this paper proposes a heuristic method that enables services to automatically achieve horizontal elastic scaling during runtime. The purpose of this method is to flexibly increase service instances based on the usage of cluster resources when user requests change, in order to improve the stability of service response, or reduce service instances to reduce the waste of computing resources.

According to the details have been introduced, the whole paper is organized as follows. Section 2 reviews related work. Section 3 introduce the multi-objective microservice deployment problem. Section 4 shows the detail of deep reinforce learning method including reinforce learning model and step of deep learning. Section 5 shows the results of the experiment on Kubernetes. Finally, the conclusion represents on section 6.

2 Related work

In recent years, there has been many research on deploying microservice in a multi-node edge computing environment.

2.1 Container based microservice

Container based microservice architecture has become a research hotspot in cloud computing due to its isolation and stability [40]. Ionescu and Diaconita [17] proposed that advanced technologies such as AI and cloud computing provide new development methods for system development. Saha et al. [34] discussed the advantages of latency reduction and performance improvement brought by distributed computing. Hoang et al. [14] pointed out that Edge computing is also meant to meet the increasing demands of the Internet of Things (IoT)/ Internet of Everything (IoE). Cai and Buyya [5] proposed a feedback control method aimed to improve the accuracy of output errors, which guarantee the QoS in Kubernetes. This feedback method has a certain level of complexity due to the combination of varying-processing-rate queuing model and linear-model. However, it didn't consider the waste of node resource. Mao et al. [25] designed a novel container scheduler for short-lived deep learning applications, which considered typical characteristics of training processes on Kubernetes. For the limited resource in edge computing, Li et al. [20] constructed the robust application deployment problem as a constrained optimization problem and proposed an approximate method for finding the approximate optimal solution. Deng et al. [9] proposed a method to minimize the costs of service deployment when resources are limited in mobile edge computing (MEC).

2.2 Microservice deployment model

Gao et al. [12] proposed a unified scheduler that optimized multiple scheduling objectives in a GPU cluster, which including Estimator for estimating the job duration, and Selector for selecting jobs and allocating resources. Peng et al. [30] proposed a DL-driven scheduler for deep learning cluster, which including the offline neural network to warm up and the online neural network to solve the problem. Duan et al. [11] considered a multi-objective optimization problem on the low Earth orbit satellite environment, which included limited computing resources and service latency. However, it has certain limitations due to the lack of simulation experiments on Kubernetes. Deng et al. [8] proposed high-quality placement and low training complexity, but cannot cover the complex interactions between containers under constraints from a global perspective. Wang [43] proposed

the hybrid scheduling of cloud computing resources, including swarm optimization algorithm-based resource scheduling model. These papers all have some issues, some of them do not effectively combine cluster resource optimization with user service requests, and some have limitations in implementing the environment. Therefore, this article will fully consider the different user requests during high concurrency and dynamically allocate cluster computing resources to ensure service response latency and cluster resource balance.

2.3 Optimization of deep reinforcement learning

The deep reinforcement learning algorithm has been widely applied for strategic decision-making problem, where the agent learns a policy to optimize a cumulative reward by trial-and-error interactions with the environment [47]. Hu et al. [15] designed a multi-objective optimization problem that combines request routing and microservice deployment, using deep reinforcement learning methods to construct a model and solve it, aiming to reduce resource consumption, request latency, and energy waste. Chen et al. [6] designed a model based on graph attention network to address the heterogeneity and connectivity of edge nodes, and combined it with deep reinforcement learning to solve the optimal microservice deployment strategy. Lv et al. [21] used the RSDQL algorithm for multi-node deployment of microservice chains. They also proposed a heuristic-based horizontal scaling algorithm, which could not address the routing problem for multi-instance deployment. Jiang et al. [18] proposed a new edge computation-based framework for IoT data processing and scheduling using deep reinforcement learning.

Huang et al. [16] and Burns et al. [4] proposed the scheduling optimization algorithm based on reinforcement learning on Kubernetes. The purpose is to maximize resource utilization and minimize scheduling time under the constraints of CPU and memory resources, and the effectiveness of the method has been verified through comparison. But their reinforcement learning methods have drawbacks in reward settings and execution time has not been optimized.

The implementation of these deep reinforcement learning methods has demonstrated their superior performance compared to traditional deployment methods. However, most existing DRL methods consider single-objective optimization; few address multi-objective optimization with automatic scaling under high concurrency.

Therefore, in the environment of high concurrency user requests and dynamic resource allocation, this article designs a multi-objective optimization problem with low service latency and balanced cluster resources. Deep reinforcement learning method is used to obtain the optimal deployment strategy, and a heuristic algorithm is designed to achieve automatic service scaling and ensure service stability. In addition, we conduct the entire algorithm on Kubernetes, which has reference significance for microservice development.

3 System model and problem formulation

The purpose of this paper is to ensure low service latency, service stability and cluster resource balance in the edge computing environment with high concurrent access and dynamic resource allocation. Therefore, in this section, we provide a detailed description of the design process for the system model, user request model, load balancing model, and multi-objective optimization model. The system model provides an overall and comprehensive system description, the user request model describes service latency based on user preferences, the load balancing model describes the balance of all computing resources on all edge nodes combined horizontal scaling, and the multi-objective optimization model includes the above optimization objectives. There is summary of main notations showed in Table 1.

3.1 System model

A set of nodes owned by a complete microservice cluster can be expressed as $N = \{N_1, N_2, \cdots, N_i, \cdots N_{|N|}\}$, where N_i represents a physical node in the cluster. $|N|$ indicates the total number of edge nodes, the subscript $1 \sim N$ indicates the number of each edge node. In each edge node, there are several computing resources with different capacities, such as CPU, memory, IO, and network

Table 1: Summary of main notation

Notations	Definition
N	the set of edge nodes
R	the set of computing resources
R_{ij}	the available capacity of resource j on node i
UR	the set of user requests
UM	the set of microservices for a user request
M	the set of containers for a microservice
r_{kj}	the required of resource j for container k
x_{si}	the user request s running on node i
n_{ki}^{l}	the number of containers on node i
y_{ki}^{l}	the number of microservices on node i
V	the set of variances for resources
λ	the weight of resources
$T_{\cos}^{s}$	the delay of microservice running
T_{que}^{s}	the delay of requests chain
T_{tran}^{s}	the delay of transmission of containers
T	the total delay of user request
S	the state of agents
RW	the reward in training
ST	the action of agents

transmission rate. These resources owned by each edge node n_i are represented as a set of $R = \left\{R_{i1}, R_{i2}, \cdots, R_{ij}, \cdots R_{i|R|}\right\}$, where $|R|$ denotes the number of resources on each node and R_{ij} denotes the available capacity of resource j on node i.

In this paper, we consider user requests under high concurrency situation. We design a set of $UR = \left\{UR_1, UR_2, \cdots, UR_S, \cdots UR_{|UR|}\right\}$, where UR_s denotes one of user requests and $|UR|$ denotes the number of user requests at the same time. When a user request comes, the corresponding microservices will be deployed and run in the cluster. In addition, a user request may require multiple microservices in order to receive corresponding service response. Therefore, we design a set of $UM = \left\{UM_{s1}, UM_{s2}, \cdots, UM_{sl}, \cdots UM_{s|UM|}\right\}$, where UM_{sl} denotes the microservice l deploying for the user request s and $|UM|$ denotes the number of microservice running for service response. In order to ensure the stability of microservice, we need to design the horizontal scalability of microservice instances. Therefore, we assume that containers of a microservice in the cluster could be expressed as a set of $M = \left\{M_{l1}, M_{l2}, \cdots, M_{lk}, \cdots M_{l|M|}\right\}$, where M_{lk} denotes the container k of microservice l and $|M|$ denotes the number of containers. Therefore, the required resources of container k are represented as a set of $r = \left\{r_{k1}, r_{k2}, \cdots, r_{kj}, \cdots r_{k|R|}\right\}$, where r_{kj} denotes the capacity of required resource j. For user request s, we design x_{si} denotes the number of microservices running for user request s on the node i. When $x_{si} = 0$ means the microservices running for user request s didn't deploy on the physical edge node i, otherwise $x_{si} = 1$. However, for user request s, there may not necessarily be only one microservice l running for user request s on the same node i. We design y_{li}^{s} which denotes the number of microservices for user request s on node i. In addition, for microservice l, there also may not necessarily be only one container k on the same node i. we either design n_{ki}^{l} which denotes the number of containers running on node i. The corresponding relationship between such system variables showed in Fig. 1.

According to the system model mentioned above, the relevant parameters need to have constraints as Eqs. (1) and (2). Eq. (1) indicates that the sum of resources consumed by all containers running on each physical edge node must be within the corresponding resource capacity range of the node.

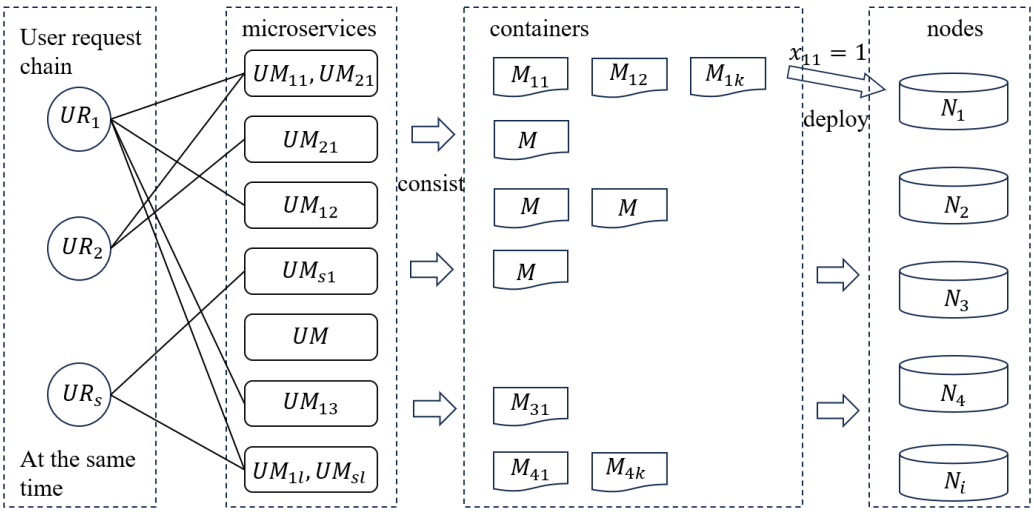


Figure 1: System Model Showing Relationships Between User Requests, Microservices, Containers, and Edge Nodes.

$$\sum_{s=1}^{|UR|}\sum_{l=1}^{|UM|}\sum_{k=1}^{|M|} r_{kj}x_{si}n_{ki}^{l}y_{li}^{s} \leq R_{ji} \tag{1}$$

Eq. (2) indicates that there are only two scenarios for running microservices on physical edge nodes, and its values are specified.

$$\begin{cases} x_{si}=1, y_{li}^{s}=1, n_{ki}^{l}>0 \\ x_{si}=0, y_{li}^{s}=0, n_{ki}^{l}\leq 0 \end{cases} \tag{2}$$

Generally, quality of service (QoS) needs to be considered in edge computing based on microservices. Especially in environments with high concurrency user requests, low service latency is crucial for QoS. In this paper, referring to the work [9, 46], we consider influence factor of service latency based on requests queuing delay, microservice running delay, data transmission delay. According to the microservice request scheduling referring to latest work, we model the request process as a queue network model.

Microservice running delay: For a user request in the above user request queue, its service response may be run by a combination of multiple microservices, and each microservice may be run by a combination of multiple containers. Therefore, its service response time is mainly composed of the data transmission delay between containers and the running delay of different containers. Here we first provide the microservice running delay that affects the service response results. We assume that when the required resources of a container are met, its running time at different physical edge nodes is consistent. We set t_{con}^{l} as the consumption time for container operation and T_{cos}^{S} as consumption time for a user request on total physical edge nodes. So T_{cos}^{S} is calculated as Eq. (3).

$$T_{cos}^{s} = \sum_{i=1}^{|N|}\sum_{l=1}^{|UM|}\sum_{k=1}^{|M|} t_{con}^{l}x_{si}n_{ki}^{l} \tag{3}$$

Requests queuing delay: When multiple user requests occur at the same time, there must be queued for processing. According to the existing work [13], we apply the M/M/1 queuing network theory to handle this type of request queuing delay. We set μ_{n}^{m} denotes the service rate of microservice instances running on different edge nodes, which is calculating as Eq. (4).

$$\mu_n^m = \frac{CR_n}{CI_m} \tag{4}$$

In the Eq. (4), CR_n denotes the computational rate of physical edge node n and CI_m denotes the computational intensity of microservice m.

When a single microservice instance belongs to microservice chains, the total traffic is expressed as Eq. (5).

$$\theta_{n_i}^{m_l} = \sum_{l=1}^{|UM|} x_{n_i}^{m_l} \theta_{UM_l}^{n_i} \tag{5}$$

where $\theta_{m_l}^{n_i}$ denotes the flow size of microservice chain on the edge nodes.

According to the queue method, t_{que}^l denotes the delay of the microservice m deploy on the node n, which is calculated as Eq. (6).

$$t_{que}^l = \frac{\theta_n^m}{\mu_n^m \left(\mu_n^m - \theta_n^m\right)} \tag{6}$$

The total queuing delay of the microservice m in the microservice request chain at multiple instances of multiple edge nodes is represents as Eq. (7).

$$t_{m_l}^{que} = \sum_{i=1}^{|N|} x_{n_i}^{m_l} t_{que}^l \tag{7}$$

Therefore, the total average queuing delay T_{que}^l of the microservice request chain MC is calculated as Eq. (8).

$$T_{que}^s = \sum_{l=1}^{|UM|} t_{m_l}^{que} \tag{8}$$

Data transmission delay: For the data transmission delay, it is not only necessary to consider the transmission between physical edge nodes, but also the transmission between different containers on the same node. Therefore, for the deployment of different microservices on different physical edge nodes, we denote the transmission delay time between physical edge nodes is t_{NodeT}, which is calculated as Eq. (9).

$$t_{nodeT} = \frac{D_k}{T_i} \tag{9}$$

where D_k is the total amount of data that the container needs to transmit and T_i is the transmission rate between nodes. then, we denote the transmission delay time between different containers on the same edge node is t_{ConT}, which is calculated as Eq. (10).

$$t_{ConT} = \frac{D_k}{t_i} \tag{10}$$

where t_i is the transmission rate of node i. So, the data transmission delay of a user request T_{tran}^s, which is calculated as Eq. (11).

$$T_{tran}^s = \sum_{i=1}^{|N|} x_{n_i}^{m_l} \left(t_{nodeT} + t_{ConT}\right) \tag{11}$$

The total delay of a user request response T, which is calculated as Eq. (12).

$$T = T^s_{cos} + T^s_{que} + T^s_{tran} \tag{12}$$

The lower the total delay time, the shorter the service response time.

3.2 Load balance model

In a microservice cluster, the resource capacity of each edge node is different, and the resources required for different microservices are also different. Therefore, the goal of cluster load balancing is to allocate corresponding computing resources to corresponding microservices in a reasonable manner, which helps to reduce resource waste and redundancy in the cluster and correspondingly improve resource utilization.

In the microservice cluster, when $x_{si} = 1$, it demotes container k running on node i corresponding to microservice l. Therefore, a set of $U = \left\{U_{j1}, U_{j2} \cdots, U_{ji}, \cdots U_{j|N|}\right\}$ is denoted as utilization rate of resources in the cluster, where U_{ji} denotes the resource utilization of resource j on the node i. So U_{ji} is calculated as Eq. (13).

$$U_{ji} = \frac{\sum_{s \in UR} \sum_{l=1}^{|UM|} \sum_{k=1}^{|M|} r_{kj} x_{si} n^l_{ki} y^s_{li}}{R_{ji}} \tag{13}$$

We use $\bar{U}_j$ represent the average utilization rate of resource j on all nodes, which calculated as Eq. (14).

$$\bar{U}_j = \frac{\sum_{i=1}^{|N|} U_{ji}}{|N|} \tag{14}$$

The variance of all resources can be defined as $V = \left\{V_1, V_2, \cdots V_j, \cdots, V_{|R|}\right\}$. In the microservice cluster, we consider four types of computing resources including CPU, memory, IO and network transmission rate. Therefore, the variance of resource is calculated as Eq. (15).

$$V_{cpu} = \frac{1}{|N|} \sum_{i=1}^{|N|} \left(U_{ji} - \bar{U}_{ji}\right)^2, j = cpu \tag{15}$$

The variance of memory resource is calculated as Eq. (16).

$$V_{mem} = \frac{1}{|N|} \sum_{i=1}^{|N|} \left(U_{ji} - \bar{U}_{ji}\right)^2, j = mem \tag{16}$$

The variance of IO resource is calculated as Eq. (17).

$$V_{IO} = \frac{1}{|N|} \sum_{i=1}^{|N|} \left(U_{ji} - \bar{U}_{ji}\right)^2, j = IO \tag{17}$$

The variance of network transmission rate resource is calculated as Eq. (18).

$$V_{ntr} = \frac{1}{|N|} \sum_{i=1}^{|N|} \left(U_{ji} - \bar{U}_{ji}\right)^2, j = ntr \tag{18}$$

Due to the irregular demand for resources in microservices and the varying emphasis on different resources, this article designs resource weights $\lambda = \left(\lambda_1, \lambda_2, \cdots, \lambda_{|R|}\right)$ to better balance the load of the cluster.

Then, we can calculate the weighted variance of resources is

$$V_{useage} = \sum_{j=1}^{|R|} \lambda_j \times V_j^T \tag{19}$$

Therefore, Lower variance means resources are more evenly balanced across nodes, improving stability.

3.3 Multi-objective model

As the detailed description above, the multi-objective optimization model for microservice deployment and user requests under high concurrency environment, as shown in Eq. (20). The model consists of two parts, including multiple optimization objectives and constraint conditions. The objective of the model is to minimize load balancing and service latency. We consider capacity computing resource including CPU, memory, IO and network transmission rate. The service response mainly including requests queue delay, microservice running delay, data transmission delay.

$$\begin{cases} \min V_{useage} \\ \min T \\ \text{s.t. } (1), (2) \end{cases} \tag{20}$$

We name the above microservice deployment and user requests problem as a Multi-Objective Microservice Deployment Problem (MMDP). The multi-objective optimization problem proposed in this article can be regarded as a mixed linear programming problem due to the complexity of its model and the dynamic nature of its constraints [3, 24]. We propose a method that combines container adaptive horizontal scaling and optimal microservice deployment strategy to provide stable and low latency service responses under high concurrency user requests and load balancing environments.

4 Design and implementation

In order to solve the problem of optimal service deployment and low service latency under high concurrency situations as mentioned above, we propose the hybrid method combined with container automatic horizontal scaling and optimal microservice deployment. In this section, we first use deep reinforcement learning method (DRFLM) to construct the multi-objective microservice deployment problem as mentioned above, and combine deep learning method to solve it for obtaining the optimal microservice deployment strategy under dynamic edge computing environment. The RFLMD method in this paper is based on the DQN algorithm, introduces reinforcement learning to handle multi-objective optimization problems, and designs rewards for optimization, thereby obtaining the optimal results. Afterwards, we design a heuristic horizontal container auto scaling method to obtain the number of containers that meet the requirements for service operation. According to mixed method, we can achieve more stable service response, shorter service latency, and more balanced cluster resources.

4.1 DRFLM for microservice deployment

The purpose of reinforcement learning is to solve optimization problems and obtain optimal optimization strategies while satisfying dynamic constraints. It is based on Markov theory to design the model state, action, and reward, so that the model can obtain the optimal result that satisfies the reward through training. Therefore, reinforcement learning is suitable for decision-making problem, such as dynamic task scheduling, where the optimal strategy is achieved through a series of actions [44].

In this article, the multi-objective microservice deployment model is a step-by-step serialization of actions taken under dynamic constraints and states to maximize expected returns. Therefore, the learning ability of reinforcement learning methods is applicable to the research of microservice deployment strategies.

In this section, to address the aforementioned multi-objective microservice optimization problem, we propose a Markov reinforcement learning method based on reward shaping. This method mainly consists of two parts. One is to train the agent offline using the deep reinforcement learning method, and the other is to obtain the optimal deployment strategy online based on the real state by using the trained agent as the deployment controller. In reinforcement learning model training, the agent obtains input from the current states S_t, calculates the deployment operation to be performed based on the above multi-objective model that satisfies resource balance and has low time consumption, and then calculates according to the reward function based on the current state to obtain a reward value to evaluate the performance of the action, and then transitions to the next state S_{t+1} step by step. After completing the deployment trajectory path of the entire user request, the cumulative reward is calculated to evaluate the advantages and disadvantages of the status. Then, after the agent completes the deployment plan of the entire user request, the entire model is updated. Through continuous iterative training, the agent acquires the optimal action and deployment strategy with the greatest return in the current environment. After obtaining the real data and status, the trained agent selects the best deployment plan for the deployment controller in the real environment based on the current status. The entire process mentioned above is shown in Fig. 2.

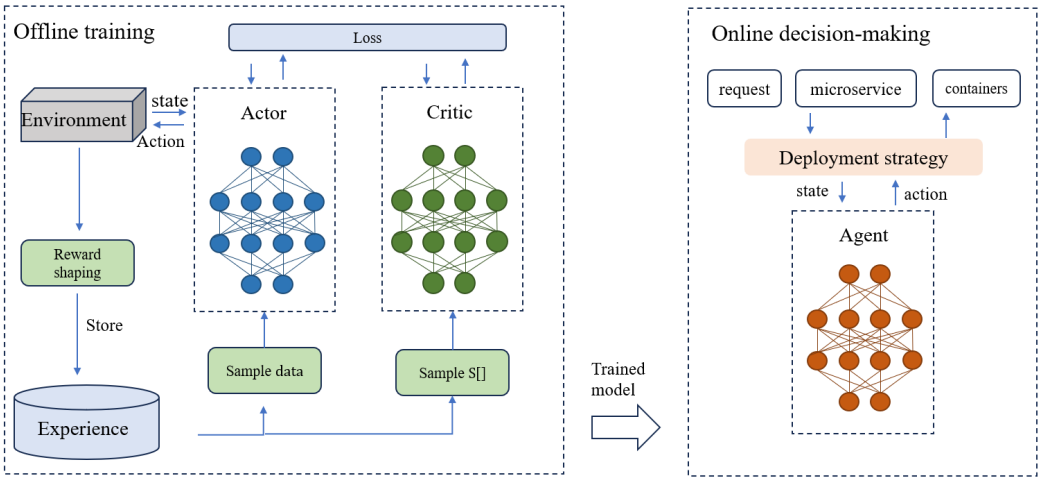


Figure 2: The Architecture and Operation Process of DRFLM Method Including Agents, Actions and Rewards.

The main factors of Markov method including states, agents, actions, and rewards. So, we provide a detailed description of four key elements referring to multi-objective problem.

Status: According to the previous system model, the state variables in Markov theory in this article mainly include node state, container state, and user request state. We set S denote the status, which is calculated as Eq. (21).

$$S = \{S_{Node}, S_{Con}, S_{User}\} \tag{21}$$

The $S_{Node} = \{NodeST, CPUST, MemST, DTRST, NTRST\}$ denotes the status of edge nodes , which NodeST including number of edge nodes, CPUST denotes available capacity of CPU resources on edge nodes, MemST denotes available capacity of memory resources on edge nodes, DTRST denotes available capacity of device transmission rate resources on edge nodes, NTRST denotes available capacity of network transmission rate resources on edge nodes. The $S_{Con} = \{ConST, ReqST\}$ denotes the status of containers on edge nodes, which ConST denotes the number of containers for corresponding microservice on each edge nodes. And ReqST denotes the required resources

of containers including CPU, memory, device transmission rate, network transmission rate. The $S_{User} = \{URST, MISST, MaxST\}$ denotes the status of user requests, which URST denotes the number of user requests at the same time, MISST denotes the number of microservice of a user request, MaxST denotes the maximum tolerable service requests delay.

Agents: The agent represents the new action and status of microservice deployment, which are used to update the parameters in edge cluster.

Actions: The action of the DRFLM is deploy containers on edge nodes satisfying the constraints (1) and (2). In addition, a user request may require more than one microservice to provide a service response, and a microservice may deploy multiple corresponding containers on multiple edge nodes. So, under the premise of determining the required number of microservices for each user request, finding the appropriate number of container deployments is necessary to achieve the lowest resource load consumption and service response latency in the cluster.

We design a $M \times |N|$ matrix to represent the possibility of deploying microservice to node.

$$\begin{bmatrix} x_{11} & \cdots & x_{1|N|} \\ \vdots & x_{si} & \vdots \\ x_{M1} & \cdots & x_{M|N|} \end{bmatrix}$$

According to the matrix, if a user request requires three microservices to implement service response. To meet the service latency conditions, assume that microservice 1 needs to run two containers. When one container deploying on node $n_1, x_{11} = 1$. The other container deployment on node $n_2, x_{12} = 1$.

Rewards: In reinforcement learning theory, the reward function determines whether the model can obtain better optimization strategies through training. According to the deployment problem. We need to train the model step by step to solve multi-objective optimization problems, in order to obtain a complete optimal deployment strategy. In this process, using Markov theory, rewards need to be set for each action in response to deployment problems. In addition, we can only evaluate the deployment strategy after all user requests have been deployed, which means we cannot evaluate the model in each action.

We propose reward shaping into reinforcement learning training referring to the work [47]. We propose a reinforcement learning method based on reward accumulation. In each training session, the agent executes the complete microservice deployment as the default action, meaning that each step of the agent's execution is a complete action. Punish the agent appropriately if the chosen action violates the constraint conditions. After completing all the training of the agent, calculate the cumulative maximum reward for the model and provide feedback, ultimately obtaining the optimal deployment strategy. The reward function is setting as follow Eq. (22).

$$RW = \begin{cases} -\omega_1 T^s_{que} & \text{if } i = 1 \\ c_2 - \omega_1 T - \omega_2 V & \text{if } i = |UM| - 1 \\ -\omega_1 \left(T^s_{que} + T^s_{\text{trans}} \right) - \omega_2 V & \text{otherwise} \\ -\dfrac{\theta^i_l}{\sum_{l=1}^{|U|'} \theta^i_l} & \text{loss} \end{cases} \tag{22}$$

where ω_1, ω_2 denotes the weight of reward function and c_2 is the positive bias reward when the agent has successfully deployed a microservice instance.

4.2 Heuristic horizontal container auto scaling method

Although we obtained the optimal microservice deployment strategy through the aforementioned DRFLM method which is satisfied the conditions of cluster resource balance and low service response. There may be the situation that the resource should be limited when a certain amount of microservices reached on a node, or when a certain microservice is heavily accessed. At this point, in order to ensure quality of service, it is necessary to adjust the number of containers and deployment of microservices, appropriately increasing the number of containers or deploying them to other nodes. Therefore,

we design a heuristic container horizontal scaling method to adaptively adjust the such situation of microservices and nodes.

When we get the user request, the corresponding microservice chain M_l for service response is considered. We consider the computing resource pressure of each edge nodes represented as P_i. The maximum response delay of the user request is t_l. The weight of deployment on node is σ_i. Firstly, we evaluate the possibility of accessing microservices and calculate the available resource pressure of the node where the microservice deployment container is located. Secondly, based on the maximum response delay of user requests, calculate the minimum number of microservice containers within the delay requirement; And based on the required resources of the container, calculate the maximum number of microservice containers while meeting the requirements of node resource balance. Finally, based on the number of containers obtained and the resource pressure of different nodes, we consider the capacity resources of each edge nodes, and deploy the containers into different edge nodes.

5 Experiment

In this section, we introduce three parts of experiment, which including algorithm application scenarios, experimental parameter settings, and comparison of experimental results.

5.1 Scenarios

For the experiment of the methods, we design and conduct an industrial tool management platform based on real tool data and management requirements on Kubernetes. According to the actual situation of the tool management platform, the physical servers of the platform are distributed in different areas such as the machining workshop and the management control center. In order to achieve efficient use of resources and distributed data management, we propose the design and development of the tool management platform based on edge computing and microservice architecture. We virtualize the physical servers of the platform into multiple edge nodes, virtualize physical resources into service data, and split the platform's related service functions into multiple microservices, which are deployed on different edge nodes. The microservices exchange data through interfaces to achieve isolation between services. Through microservice architecture, not only has the utilization efficiency of computing resources in various physical servers been improved, but the platform has also gained stronger flexibility, scalability, and stability. The scenarios are showed as Fig. 3.

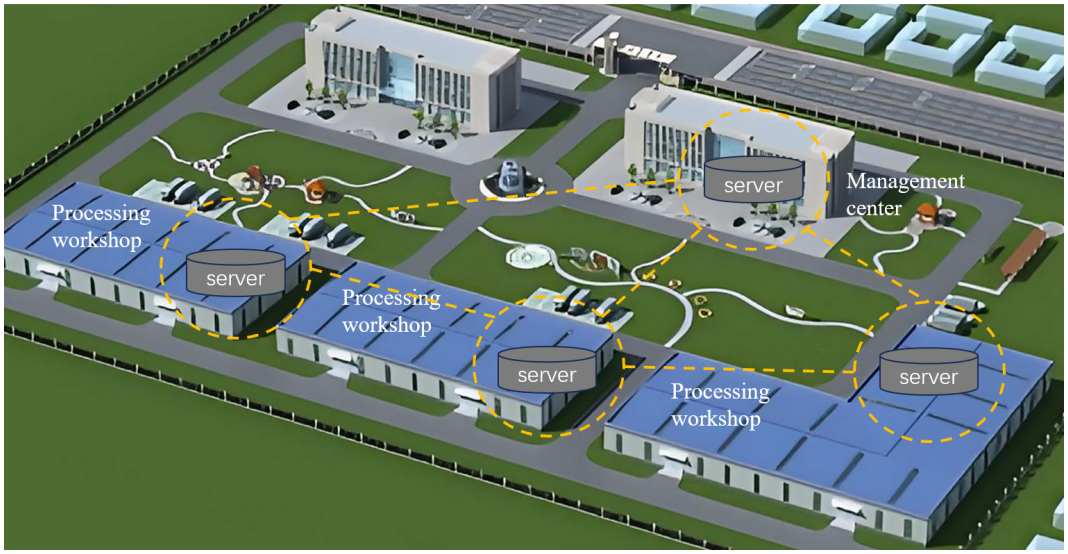


Figure 3: Scene of Tool Management Platform.

We conduct the experiments on such tool management cloud service platform. According to the design scheme of microservice architecture and edge computing, we built the above platform on Kubernetes [41] and Alibaba Spring Cloud. We develop microservices separated from the platform

in Spring Cloud, package them into images, and store them in Kubernetes' image repository. When calling microservices, generate and deploy corresponding Docker containers in Kubernetes based on the corresponding images, and communicate between containers through API interfaces.

The powerful orchestration and management capabilities of Kubernetes allow us to individually edit the calling and scaling methods of containers, facilitating the experiments we propose.

5.2 Experiment settings

The tool management platform based on Kubernetes consists of five nodes, which including one master node and four working nodes. The configuration of the master node includes an Intel i9-13900k CPU and an RTX3090 GPU, which has fast enough processing speed for the DRFLM algorithm. We provide the platform details in Table 2, which including computing resources, the version of Kubernetes on each edge nodes in cluster.

Table 2: Capable resources of edge nodes on Kubernetes

Node	CPU	Memory	IO	Trans of Net
Master	30cores	32G	500MB/s	100MB/s
Working1	20cores	32G	500MB/s	100MB/s
Working2	20cores	32G	500MB/s	100MB/s
Working3	20cores	32G	500MB/s	100MB/s
Working4	20cores	32G	500MB/s	100MB/s

In the design of the platform, we use the Prometheus as a cluster monitoring tool [26]. Its open-source nature and powerful real-time monitoring of various resources make it easy to deploy in Kubernetes. Therefore, we use it to monitor the usage of computing resources on clusters and containers during the deployment and running process. In addition, our user requests are based on the actual needs and data of the tool management platform. According to the design of the platform management, we provide at least ten microservice images, so that when users make requests, corresponding containers can be deployed and corresponding service responses can be obtained. In Table 3, we list the required resources of the microservices.

Table 3: Required resources of microservices in cluster

Microservice	Memory	CPU	IO	Trans of Net
Microservice1	0.5core	256MB	10MB/s	1MB/s
Microservice1	0.5core	128MB	20MB/s	512KB/s
Microservice1	0.8core	512MB	25MB/s	1.5MB/s
Microservice1	0.5core	256MB	20MB/s	1MB/s
Microservice1	0.9core	256MB	15MB/s	2MB/s
Microservice1	1.0core	512MB	25MB/s	2.5MB/s
Microservice1	0.3core	128MB	5MB/s	512KB/s
Microservice1	0.2core	128MB	5MB/s	512KB/s
Microservice1	0.6core	256MB	15MB/s	1.5MB/s
Microservice1	0.7core	256MB	20MB/s	2MB/s

We take experiments of the other three methods compared with our method, which including Kubernetes Default Scheduling method, DQL method, and FFD method. FFD is a first-fit greedy algorithm that has been widely used for service placement [29]. All these methods are implemented on master node and obtained the same user requests. Therefore, we compare the results in five aspects, which including response time, load balancing, results of multi-objective microservice optimal deployment, convergency and results of edge nodes increasing.

Due to the increased flexibility of Kubernetes scheduling algorithms with the development of microservices, we can replace scheduling methods through plugins. In this article, during the experimental process, we separately developed the model and method as a scheduling tool, replacing the

default scheduling method as a plugin to achieve real-time container scheduling. And combined with Prometheus, achieve real-time scheduling and management of the platform.

5.3 Experiment Results

Under the same service request, we deployed 40 containers in the cluster and compared the response delay, variance and convergence data of the above four methods as shown in Table 4. According to the data in the table, it can be seen that compared with the other three methods, DRFLM has lower response delay, lower variance and better convergence. This indicates that our method responds faster, has more balanced resources, and is more robust.

Table 4: Capable resources of edge nodes on Kubernetes

Method	Number of containers	Response time	Variances of function	Convergence
DRFLM	40	356ms	141.34	-1
DQL	40	501ms	159.64	-7
FFD	40	630ms	174.35	-
Default	40	690ms	185.43	-

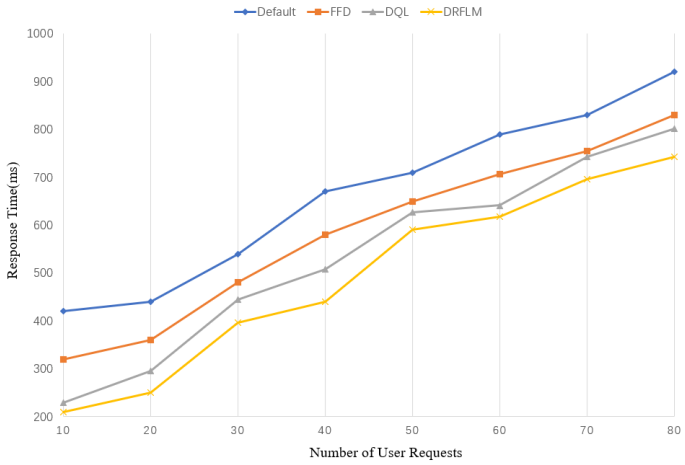


Figure 4: The Response Time of User Requests.

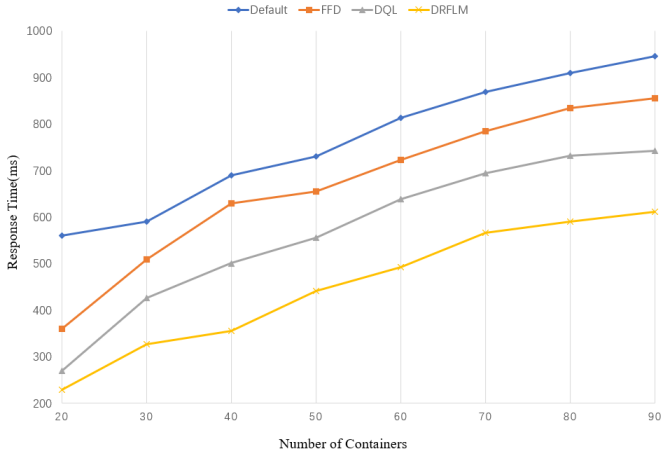


Figure 5: Response Time of User Requests: DRFLM reduces latency by 25% Compared with DQL and 30% Compared with Default.

Response delay: For the response delay of user requests, Fig. 4 shows the average service response time of three algorithms on Kubernetes for the same chain of user requests. And Fig. 5 shows the

average service response time of containers increasing in cluster. According to the user request model mentioned above, the service response time mainly combine three parts. As the experiment results, the microservice running delay are similarly based on DRFLM method, DQL method, FFD method and default method. But the data transmission delay and the requests queue delay of the user requests are better based on such four methods. It is because of the different optimal deployment strategy achieved by different method. The corresponding containers are deployed on different nodes. It leads to the different containers running for user requests.

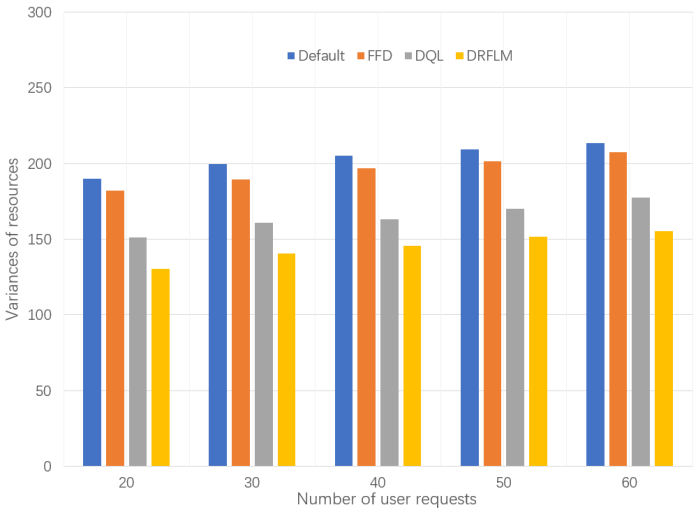


Figure 6: The Resources Variances Based on Different User Requests

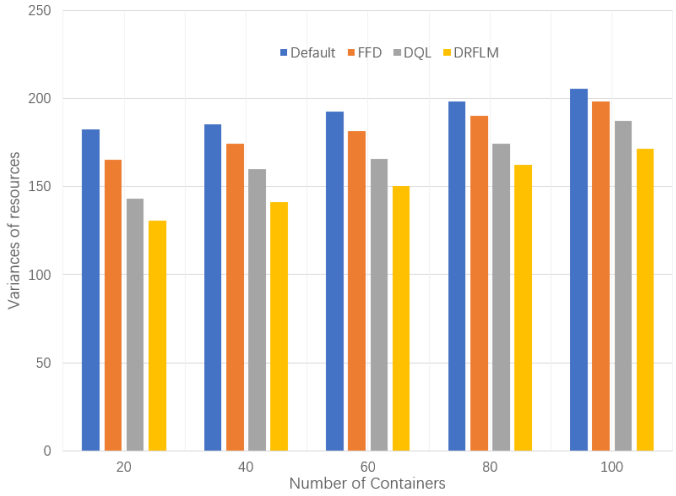


Figure 7: The Resources Variance Based on Containers: DRFLM Reduces Variances by 10% Compared with DQL and 17% Compared with Default.

Load balancing: Unlike the article [14, 21], our microservice for tool management in this article also has high requirements for the transmission rate on edge nodes. Therefore, the load balancing of computing resources in Kubernetes clusters includes CPU, memory, IO, and network transmission rate. We initiated user requests simultaneously for four methods and obtained the average occupancy rates of thus resources for each method, as shown in Fig. 6. In Fig. 7, the variances of resources based on different containers in cluster are shown. It can be seen that the average resource occupancy rates of the four methods are relatively low. However, Fig. 6 shows the variance of four method after implement, it can be seen that the variance of our DRFLM method is the smallest, indicating that our method can provide better load balancing. And DRFLM achieves better stability under increased

nodes, demonstrating robustness as a dynamic controller.

Results of multi-objective microservice optimal deployment: We present the results of the multi-objective microservice optimization problem solved by Default method, DRFLM, DQL, and FFD methods. Our multi-objective optimization includes minimum load balancing and minimum service latency, with the aim of ensuring average resource utilization and fastest service response. As shown in Figs. 8 and 9, it can be seen that all four methods achieved the optimization objectives mentioned above during the model training phase. Through model training, the decision strategies generated by the four methods all belong to the optimal deployment strategy, but their generation time can be clearly seen from the graph. The RFLMD method in this article is approximately 50% faster than the default method and about 20% faster than the DQL method in terms of the average policy generation time. This is because in the training and decision-making process, the RFLMD method's design of rewards is more in line with Kubernetes' deployment process.

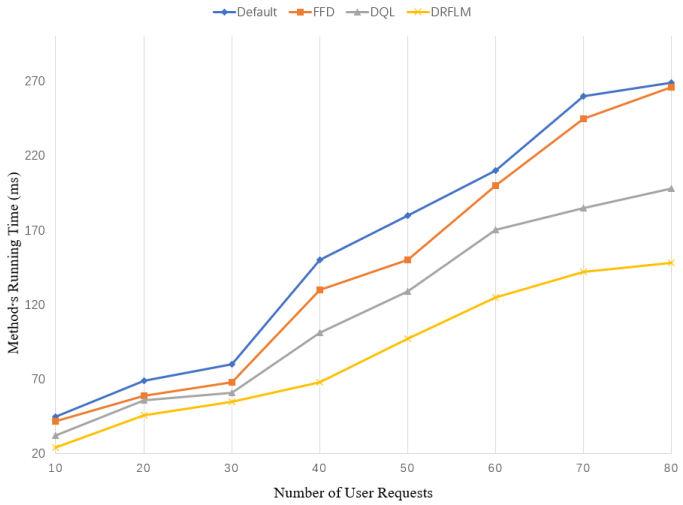


Figure 8: Using Time of Methods Conducting

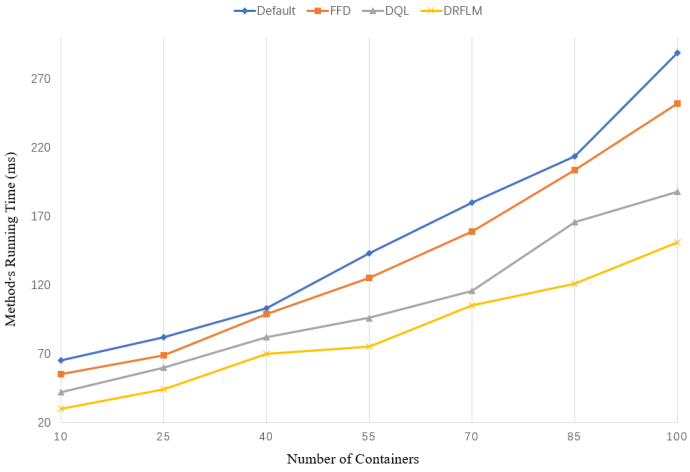


Figure 9: The Using Time of Methods Conducting: RFLMD Method in this Article is Approximately 50% Faster than the Default Method and about 20% Faster than the DQL Method.

Convergence: We conduct convergence experiments with both DRFLM and DQL at the optimal learning rate. Fig. 10 shows the convergence curves in the normalized reward training process of two methods. It can be seen that as the number of training steps increases, both algorithms are effective in finding a better strategy. It is obvious that the DRFLM algorithm converges quickly and stably within the interval of the optimal strategy compared to DQL. This indicates that in the real-time deployment of microservices, our method is more real-time and faster compared to DQL and the default method.

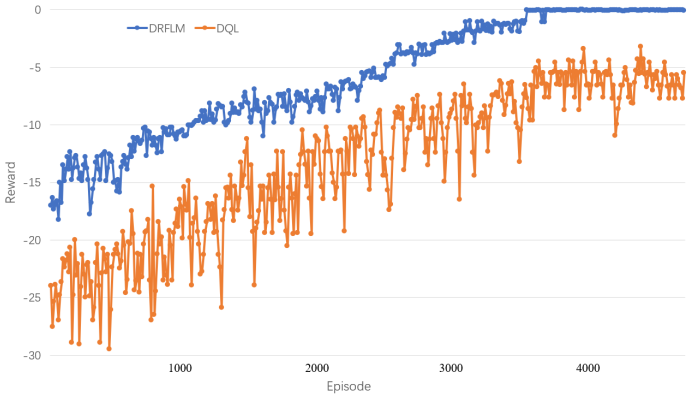


Figure 10: The Reward of DRFLM and DQL: DRFLM Achieve Better Performance on Convergence and Stability.

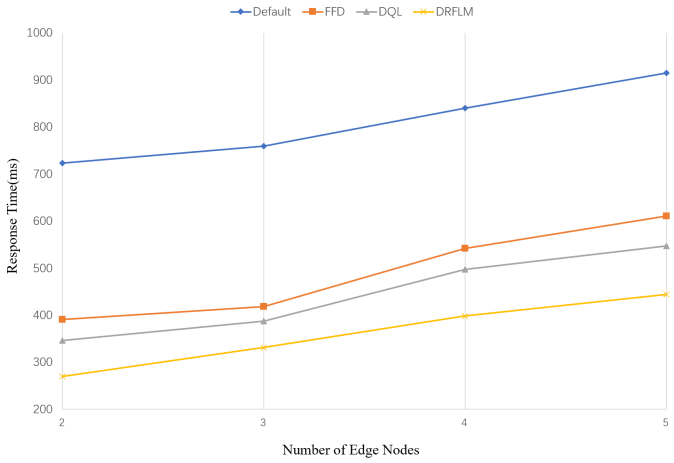


Figure 11: The Response Time of Nodes Increasing.

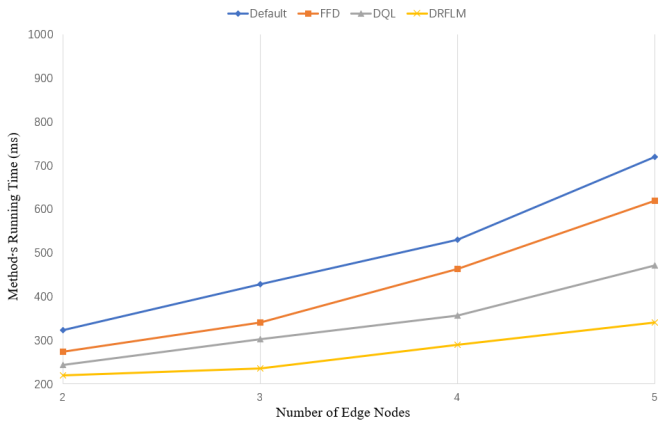


Figure 12: Method's Running Time of Nodes Increasing.

Impact of the number of edge nodes: The number of edge nodes determines the complexity of the network topology. The increase of the number of edge nodes has also increased the difficulty of microservice deployment. The results are shown in Figs. 11 and 12. As the number of edge nodes increases, the combination metric of baseline algorithms tends to decrease, and user request latency increases. the DRFLM algorithm exhibits a very gentle trend, proving the effectiveness of our proposed algorithm. DRFLM enables faster, more stable distributed control of microservices in real-time.

6 Conclusion

Through the deep reinforcement learning method based on multi-objective problems proposed in this paper for the online deployment operation of microservices, compared with the default method of Kubernetes and the DQL method, this method has a faster response time and better load balancing. This method enables the deployment and application of microservices in distributed computing to have higher efficiency, and has more practical reference significance for reinforcement learning in solving optimization problems and control problems. Future work will extend the method to multi-agent DRL across federated Kubernetes clusters and integrate communication-aware scheduling to further reduce overhead.

Conflict of interest

The authors declare no conflict of interest.

References

[1] Altaf, U.; Jayaputera, G.; Li, J.; et al. (2018). Autoscaling a defence application across the cloud using docker and kubernetes, *IEEE/ACM International Conference on Utility and Cloud Computing Companion*, Zurich, Switzerland, pp. 327–334, 2018.

[2] Brogi, A.; Forti, S.; Guerrero, C.; Lera, I. (2020). How to place your apps in the fog: State of the art and open challenges, *Software: Practice Experience*, 50(5), pp. 719–740, 2020.

[3] Burer, S.; Letchford, A.N. (2012). Non-convex mixed-integer nonlinear programming: A survey, *Surveys in Operations Research and Management science*, 17(2), pp. 97–106, 2012.

[4] Burns, B.; Grant, B.; Oppenheimer, D.; Brewer, E.; Wilkes, J. (2016). Borg, omega, and kubernetes, *Queue*, 14, pp. 70–93, 2016.

[5] Cai, Z.C.; Buyya, R. (2022). Inverse queuing model based feedback control for elastic container provisioning of web systems in Kubernetes, *IEEE Transactions on Computers*, 71(2), pp. 337–348, 2022.

[6] Chen, S.; Yuan, Q.; Li, J.; He, H.; Li, S.; Jiang, X.; Yang, J. (2024). Graph neural network aided deep reinforcement learning for microservice deployment in cooperative edge computing, *IEEE Transactions on Service and Computing*, 17(6), pp. 3742–3757, 2024.

[7] Chen, X.; Bi, Y.; Chen, X.; Zhao, H.; Cheng, N.; Li, F.; Cheng, W. (2022). Dynamic service migration and request routing for microservice in Multicell mobile-edge computing, *IEEE Internet of Things Journal*, 9(15), pp. 13126–13143, 2022.

[8] Deng, L.; Wang, Z. Y.; Sun, H.Y.; Li, B.; Yang, X. (2023). A deep reinforcement learning-based optimization method for long-running applications container deployment, *International Journal of Computers Communications & Control*, 18(4), pp. 5013, 2023.

[9] Deng, S.; Xiang, Z.; Taheri, J.; Khoshkholghi, M.A.; Yin, J.; Zomaya, A.Y.; Dustdar, S. (2021). Optimal application deployment in resource constrained distributed edges, *IEEE Transactions on Mobile Computing*, 20(5), pp. 1907–1923, 2021.

[10] Du, T.; Zhang, Y.; Shi, X.; Chen, S. (2018). Multiple slice k-space deep learning for magnetic resonance imaging reconstruction, *42nd Annual International Conference IEEE Engineering in Medicine and Biology Society*, Montreal, QC, Canada, pp. 1564–1567, 2020.

[11] Duan, H.Y.; Wu, Z.; Ji, S.; Chen, Z.; Jiang, C.X.; Zhang, W. (2025). Distributed microservice deployment for satellite edge computing networks: A multi-agent deep reinforcement learning approach, *IEEE Transactions on Vehicular Technology*, 2025.

[12] Gao, W.; Ye, Z.; Sun, P.; Zhang, T.; Wen, Y. (2024). UNISCHED: A unified scheduler for deep learning training jobs with different user demands, *IEEE Transactions on Computers*, 73(6), pp. 1500–1515, 2024.

[13] Guo, F.; Tang, B.; Tang, M. (2022). Joint optimization of delay and cost for microservice composition in mobile edge computing, *World Wide Web: Internet and Web Information Systems*, 25(5), pp. 2019–2047, 2022.

[14] Hoang, M.T.; Nguyen, N.V.; Pham, T.A.; Nguyen, T.T.; Dang, T.M.; Nguyen, H.N. (2024). Evaluating Dimensionality Reduction Methods for the Detection of Industrial IoT Attacks in Edge Computing, *International Journal of Computers Communications & Control*, 19(5), pp. 6767, 2024.

[15] Hu, M.; Wang, H.; Xu, X.; He, J.; Hu, Y.; Deng, T.; Peng, K. (2024). Joint optimization of microservice deployment and routing in edge via multi-objective deep reinforcement learning, *IEEE Transactions on Network and Service Management*, 21(6), pp. 6364–6381, 2024.

[16] Huang, J.; Xiao, C.; Wu, W. (2020). RLSK: A job scheduler for federated Kubernetes clusters based on reinforcement learning, *2020 IEEE International Conference on Cloud Engineering*, Sydney, NSW, Australia, pp. 116–123, 2020.

[17] Ionescu, S.A.; Diaconita, V. (2023). Transforming Financial Decision-Making: The Interplay of AI, Cloud Computing and Advanced Data Management Technologies, *International Journal of Computers Communications & Control*, 18(6), pp. 5735, 2023.

[18] Jiang, Y.; Wang, Z.; Jin Z.(2023). Iot Data Processing and Scheduling Based on Deep Reinforcement Learning, *International Journal of Computers Communications & Control*, 18(6), pp. 5998, 2023.

[19] Jiao, Z.; Zhang, J.; Yao, P.; Wan, L.; Ni L. (2020). Service deployment of C4ISR based on genetic simulated annealing algorithm, *IEEE Access*, 8, pp. 65498–65512, 2020.

[20] Li, B.; He, Q.; Cui, G.; Xia, X.; Chen, F.; Jin, H.; Yang, Y. (2022). READ: Robustness-oriented edge application deployment in edge computing environment, *IEEE Transactions on Services Computing*, 15(3), pp. 1746–1759, 2022.

[21] Lv, W.; Wang, Q.; Yang, P.; Ding, Y.; Yi, B.; Wang, Z.; Lin, C. (2022). Microservice deployment in edge computing based on deep Q learning, *IEEE Transactions on Parallel and Distributed Systems*, 33(11), pp. 2968–2978, 2022.

[22] Ma, X.; Yao, T.; Hu, M.; Dong, Y.; Liu, W.; Wang, F.; Liu, J. (2019). A survey on deep learning empowered IoT applications, *IEEE Access*, 7, pp. 181721–181732, 2019.

[23] Ma, X.; Zhou, A.; Zhang, S.; Wang, S. (2020). Cooperative service caching and workload scheduling in mobile edge computing, *IEEE International Conference on Computer Communications*, Toronto, ON, Canada, pp. 2076–2085, 2020.

[24] Maia, A.M.; Ghamri-Doudane, Y.; Vieira, D.; de Castro, M.F. (2019). Optimized placement of scalable IoT services in edge computing, *Proceeding IFIP/IEEE Symposium on Integrated Network and Service Management*, Arlington, VA, USA, pp. 189–197, 2019.

[25] Mao, Y.; Fu, Y.; Zheng, W.; Cheng, L.; Liu, Q.; Tao, D. (2022). Speculative container scheduling for deep learning applications in a Kubernetes cluster, *IEEE Systems Journal*, 16(3), pp. 3770–3781, 2022.

[26] Mart, O.; Negru, C.; Pop, F.; Castiglione, A. (2020). Observability in Kubernetes cluster: Automatic anomalies detection using Prometheus, *22nd IEEE International Conference on High Performance Computing and Communications*, Yanuca Island, Cuvu, Fiji, pp. 565–570, 2020.

[27] Nadareishvili, I.; Mitra, R.; McLarty, M.; Amundsen M. (2016). *Micro Service Architecture: Aligning Principles, Practices, and Culture*, Newton, MA, USA: O'Reilly Media, 2016.

[28] Pallikonda, A.K.; Bandarapalli, V.K.; Aruna, V. (2025). Enhancing performance and reducing latency in autonomous systems through edge computing for real-time data processing, *Mechatronics and Intelligent Transportation Systems*, 4(3), pp. 154–165, 2025.

[29] Peng, K.; Wang, L.; He, J.; Cai, C.; Hu, M. (2024). Joint optimization of service deployment and request routing for microservices in mobile edge computing, *IEEE Transactions on Services Computing*, 17(3), pp. 1016–1028, 2024.

[30] Peng, Y.; Bao, Y.; Chen, Y.; Wu, C.; Meng, C.; Lin, W. (2021). DL2: A deep learning-driven scheduler for deep learning clusters, *IEEE Transactions on Parallel and Distributed Systems*, 32(8), pp. 1947–1960, 2021.

[31] Poularakis, K.; Llorca, J.; Tulino, A. M.; Taylor, I.; Tassiulas, L. (2020). Service placement and request routing in MEC networks with storage, computation, and communication constraints, *IEEE/ACM Transactions on Networking*, 28(3), pp. 1047–1060, 2020.

[32] Quang, P.T.A.; Hadjadj-Aoul, Y.; Outtagarts, A. (2019). A deep reinforcement learning approach for VNF forwarding graph embedding, *IEEE Transactions on Network and Service Management*, 14(4), pp. 1318–1331, 2019.

[33] Samanta, A.; Tang, J. (2020). Dyme: Dynamic microservice scheduling in edge computing enabled IoT, *IEEE Internet of Things Journal*, 7(7), pp. 6164–6174, 2020.

[34] Saha, S.; Perumal, I.; Abbas, M.; Manimozhi, I.; Bhat, C.R. (2024). Contextual information based scheduling for service migration in mobile edge computing, *International Journal of Computers Communications & Control*, 19(3), pp. 6143, 2024.

[35] Schneider, S.; Khalili, R.; Manzoor, A.; Qarawlus, H.; Schellenberg, R.; Karl, H.; Hecker, A. (2020). Self-learning multi-objective service coordination using deep reinforcement learning, *IEEE Transactions on Network and Service Management*, 18(3), pp. 3829–3842, 2020.

[36] Schneider, S.; Qarawlus, H.; Karl, H. (2021). Distributed online service coordination using deep reinforcement learning, *IEEE 41st International Conference on Distributed Computing Systems*, DC, USA, pp. 539–549, 2021.

[37] Sheela, S.; Nataraj, K.R.; Mallikarjunaswamy, S. (2023). A comprehensive exploration of resource allocation strategies within vehicle ad-hoc networks, *Mechatronics and Intelligent Transportation Systems*, 2(3), pp. 169–190, 2023.

[38] Shi, T.; Ma, H.; Chen, G.; Hartmann, S. (2020). Location-aware and budget-constrained service deployment for composite applications in multi-cloud environment, *IEEE Transactions on Parallel and Distributed Systems*, 31(8), pp. 1954–1969, 2020.

[39] Smet, P.; Dhoedt, B.; Simoens, P. (2018). Docker layer placement for on-demand provisioning of services on edge clouds, *IEEE Transactions on Network and Service Management*, 15(3), pp. 1161–1174, 2018.

[40] Tan, B.; Ma, H.; Mei, Y. (2020). A NSGA-II-based approach for multi-objective micro-service allocation in container-based clouds, *Proc. 20th IEEE/ACM International Symposium on Cluster, Cloud, and Internet Computing*, Melbourne, VIC, Australia, pp. 282–289, 2020.

[41] Toka, L.; Dobreff, G.; Fodor, B.; Sonkoly, B. (2021). Machine learning based scaling management for Kubernetes edge clusters, *IEEE Transactions on Network and Service Management*, 18(1), pp. 958–972, 2021.

[42] Wang, S.; Guo, Y.; Zhang, N.; Yang, P.; Zhou, A.; Shen, X. (2021). Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach, *IEEE Transactions on Mobile Computing*, 20(3), pp. 939–951, 2021.

[43] Wang, Y. (2025). A Genetic Particle swarm optimization based Hybrid Scheduling Algorithm for Cloud Computing Resources, *International Journal of Computers Communications & Control*, 20(4), pp. 6814, 2025.

[44] Wang, Z.; O'Boyle, M. (2018). Machine learning in compiler optimization, *IEEE Access*, 106(11), pp. 1879–1901, 2018.

[45] Xie R.; Tang Q.; Wang Q.; Liu X.; Yu F. R.; Huang T. (2020). Satellite-terrestrial integrated edge computing networks: Architecture, challenges, and open issues, *IEEE Network*, 34(3), pp. 224–231, 2020.

[46] Xu, J.; Chen, L.; Zhou P. (2018). Joint service caching and task offloading for mobile edge computing in dense networks, *IEEE International Conference on Computer Communications*, Honolulu, HI, USA, pp. 207–215, 2018.

[47] Yu, Y.; Yang, J.; Guo, C.; Zheng, H.; He, J. (2019). Joint optimization of service request routing and instance placement in the microservice system, *Journal of Network and Computer Applications*, 147, pp. 102441, 2019.

[48] Zhang, X.; Li, Z.; Lai, C.; Zhang J. (2022). Joint edge server placement and service placement in mobile-edge computing, *IEEE Internet of Things Journal*, 9(13), pp. 11261–11274, 2022.

Journal's webpage: http://univagora.ro/jour/index.php/ijccc/

This journal is a member of, and subscribes to the principles of,
the Committee on Publication Ethics (COPE).
https://publicationethics.org/members/international-journal-computers-communications-and-control

Cite this paper as:

Feng, H.; Shi, Y.M.; Zhen, M.K. (2026). Multi-Objective DRL for Efficient Kubernetes Microservice Scheduling, *International Journal of Computers Communications & Control*, 21(5), 7222, 2026.
https://doi.org/10.15837/ijccc.2026.5.7222