Energy-Delay-Aware Joint Microservice Deployment and Request Routing With DVFS in Edge: A Reinforcement Learning Approach

Liangyuan Wang, *Graduate Student Member, IEEE*, Xudong Liu, Haonan Ding, Yi Hu, *Graduate Student Member, IEEE*, Kai Peng, and Menglan Hu

***Abstract*—The emerging microservice architecture offers opportunities for accommodating delay-sensitive applications in edge. However, such applications are computation-intensive and energy-consuming, imposing great difficulties to edge servers with limited computing resources, energy supply, and cooling capabilities. To reduce delay and energy consumption in edge, efficient microservice orchestration is necessary, but significantly challenging. Due to frequent communications among multiple microservices, service deployment and request routing are tightly-coupled, which motivates a complex joint optimization problem. When considering multi-instance modeling and fine-grained orchestration for massive microservices, the difficulty is extremely enlarged. Nevertheless, previous work failed to address the above difficulties. Also, they neglected to balance delay and energy, especially lacking dynamic energy-saving abilities. Therefore, this paper minimizes energy and delay by jointly optimizing microservice deployment and request routing via multi-instance modeling, fine-grained orchestration, and dynamic adaptation. Our queuing network model enables accurate end-to-end time analysis covering queuing, computing, and communicating delays. We then propose a delay-aware reinforcement learning algorithm, which derives the static service deployment and routing decisions. Moreover, we design an energy-aware dynamic frequency scaling algorithm, which saves energy with fluctuating request patterns. Experiment results demonstrate that our approaches significantly outperform baseline algorithms in both delay and energy consumption.**

***Index Terms*—Edge computing, queuing theory, dynamic voltage frequency scaling, service orchestration, reinforcement learning.**

Received 19 April 2024; revised 5 December 2024; accepted 18 January 2025. Date of publication 29 January 2025; date of current version 9 April 2025. This work was supported in part by the National Science and Technology Major Project of China under Grant 2022ZD0117104, in part by the National Natural Science Foundation of China under Grant 62171189, and in part by the Key Research and Development Program of Hubei Province, China under Grant 2024BAB016, Grant 2024BAB031, Grant 2023BAB074, and Grant 2022BAA038. Recommended for acceptance by X. Fu. *(Corresponding author: Kai Peng.)*

The authors are with Hubei Key Laboratory of Smart Internet Technology, School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: hust_wly@hust.edu.cn; hust_lxd@hust.edu.cn; hust_dhn@hust.edu.cn; hust_huyi@hust.edu.cn; pkhust@hust.edu.cn; humenglan@hust.edu.cn).

Digital Object Identifier 10.1109/TC.2025.3535826

I. Introduction

MICROSERVICES as an emerging computing paradigm, decouple large monolithic applications into numerous small function modules which flexibly collaborate to provide users with a variety of Internet services [1]. This decoupled architecture facilitates convenient development and maintenance of massive Internet applications, and has been widely deployed by large companies including Microsoft and Alibaba [2], [3]. As most Internet applications are delay-sensitive, one can deploy microservices at the network edge to effectively reduce service delay [4], [5].

Compared with traditional clouds, edge computing enables low-delay microservice applications [5]. However, many Internet applications are computation-intensive and energy-consuming [6], which brings great difficulties to edge servers with very limited computing resources, energy supply, and processor cooling capabilities. To reduce service delay and energy consumption in resource-constrained edge servers, efficient microservice orchestration is necessary to enable various Internet applications to local users. However, microservice orchestration for efficient delay and energy optimization at edge servers are significantly challenging and rare in previous studies.

First, efficient microservice orchestration relies on both service deployment and request routing, which are highly intertwined. In edge servers, user requests are first fed into multiple input queues and then forwarded to a series of deployed microservice instances for subsequent processing. Since multiple microservices interact with each other to collaborate, frequent data communications among the microservice instances make the deployment and routing tightly-coupled [7], [8], [9]. Accordingly, the performance of service deployment depends on request routing policies, while request routing also requires service deployment as a prerequisite. Consequently, joint optimization of the two tightly-coupled components are necessary for efficient microservice orchestration. Nevertheless, most existing work treats microservice deployment and request routing as two isolated problems [10], [11], [12]. Some studies focus on the service instance deployment problem [13], [14], while others delve into request routing [12], [15]. Although independent partitioning can simplify the problem, these studies often overlooked the strong coupling between service deployment

and request routing, consequently failing to achieve efficient service orchestration.

Second, effective service orchestration requires fine-grained problem analysis and modeling. However, previous studies did not conduct detailed analysis and optimization to fully accommodate request queuing, communicating, and computing among microservice instances, resulting in a coarse-grained orchestration approach [16], [17]. To effectively decrease end-to-end delay, the full-service process regarding queuing delay, communication delay, and computation delay should be comprehensively investigated and meticulously optimized. Moreover, a few studies have attempted to jointly optimize microservice instance deployment and request routing, yet such approaches often simply assumed only one instance exists for each service [18]. This single-instance modeling is easy to analyze, but not feasible for realistic large-scale microservices which process huge amounts of requests in parallel. In contrast, multi-instance modeling that considers the number and location of each microservice instance deployment is a more fine-grained and practical approach, but it poses greater challenges in analysis and service orchestration. This dilemma further deteriorates for massive microservice orchestration scenarios where the service deployment and request routing are tightly coupled. In this case, it is highly valuable to design multi-instance models that can accurately analyze the delay, and fine-grained orchestrate micorservice instances.

Third, most previous work focused on delay and computation cost, neglecting energy saving in edge service orchestration [19], [20]. Only a few studies considered to minimize the number of servers to reduce energy consumption [21], [22], but such work normally employed either single-instance model or course-grained optimization methods. In addition, such papers simply considered the energy cost in a similar way to the ordinary computation cost, and designed static deployment policies to minimize such costs [23], [24]. Accordingly, these approaches lack specialized design to reduce energy consumption in practical dynamic environments. In reality, user requests dynamically fluctuate, and static deployment solutions hardly match such fluctuations, tending to cause considerable wastage in computation capabilities and energy consumption. Therefore, smart designs to adapt the dynamic request patterns are highly demanded.

Motivated by the above needs and research gaps, this paper aims to minimize delay and energy in edge by jointly optimizing microservice deployment and request routing based on multi-instance modeling, fine-grained orchestration, and dynamic adaptation. To achieve this goal, several challenges are to be addressed. The first challenge lies in the joint design of tightly-coupled service deployment and request routing. As different microservices interact and collaborate to work, numerous data communications among the microservice instances make the deployment and routing closely dependent to each other. As a result, the joint design of the two components are sophisticated. Also, this difficulty will be enlarged when considering multi-instance modeling for numerous intertwined microservice instances. Third, given this complicated multi-instance joint optimization problem, it will be significantly difficult to design fine-grained approaches that are able to accurately analyze the delay and carefully orchestrate micorservices. Last, balancing delay minimization and energy saving for the above complex problem is again challenging. Specifically, effective service orchestration policies are required to adapt dynamic environments. Consequently, specialized energy saving design that can adapt the dynamic request patterns is the last challenge.

To this end, this paper contributes a set of energy and delay optimization approaches that jointly optimize microservice deployment and request routing via multi-instance modeling, fine-grained orchestration, and dynamic adaptation. The proposed algorithms consist of a static microservice orchestration algorithm based on reinforcement learning, and a dynamic energy saving algorithm based on Dynamic Voltage and Frequency Scaling (DVFS) [25]. Our algorithms first orchestrate microservices according to the peak service demands. As service demands dynamically fluctuate (no greater than the peak values), the static orchestration decisions will be dynamically modified (lowered) by the dynamic energy saving algorithm. In this way, the sever CPU frequencies are lowered to match the demands in real time, such that energy consumption is accordingly reduced. Our contributions are listed as follows:

- We establish a set of multi-instance models based on open Jackson queuing networks, which enables accurate end-to-end time analysis covering queuing delay, communication delay, and computation delay. This fine-grained model thus facilitates highly efficient microservice orchestration in terms of both delay minimization and energy saving.
- We propose a reinforcement learning approach, referred to as Delay-Aware Reward Scaling Proximal Policy Optimization algorithm (DA-RSPPO). This algorithm derives the delay optimization decisions for static service deployment and routing. Our innovative design partitions the reward function into two stages: current action reward and last-step action reward, which effectively address the so-called action space explosion problem.
- We leverage the DVFS technique to design a dynamic energy saving algorithm, referred to as Energy-Aware Dynamic Frequency Scaling (EA-DFS). This algorithm dynamically adjusts server CPU frequencies and request routing decisions, thereby adaptively minimizing energy consumption according to the fluctuating request patterns.

The paper is organized as follows: Section II discusses related work, Section III introduces mathematical models, Section IV presents the joint microservice deployment and request routing approach for static cases, Section V describes the DVFS-based energy-saving algorithm for dynamic scenarios, Section VI evaluates the proposed algorithms, and Section VII concludes the paper.

II. Related Work

A. Design and Application of Microservices

With the rapid development of optimization frameworks for multicore systems [26], [27], the design and application of

microservices have also garnered widespread attention. Petrocelli et al. proposed an integrated platform based on microservices, containers, mobile devices, and cloud services to address compute-intensive tasks [28]. Gedeon et al. introduced the concept of microservice storage, achieving efficient edge offloading [29]. Rohan Mahapatra et al. developed a prototype lightweight ML-based scheduler, dubbed Octopus, which employs a decision tree to efficiently schedule microservices on heterogeneous clusters [30]. While the aforementioned studies effectively explore the workings of microservices and their application in specific scenarios, they overlook the tight coupling between microservice instances and the diverse routing of user requests.

B. Delay Optimization

Current research focuses on minimizing service delay in Internet services. Traditionally, service provisioning and request routing were handled separately, but some researchers now optimize them together for greater efficiency.

1) Isolated Service Deployment and Request Routing: Most of the literature focused on improving service responsiveness by optimizing service deployment. Luo et al. [10] introduced a heuristic algorithm, ParaSFC, based on the Viterbi dynamic programming algorithm to achieve parallel Service Function Chain (SFC) deployment. Fu et al. [31] proposed a dynamic resource scaling framework based on streaming data analysis to address the service placement problem. Lv et al. [11] proposed Reward Sharing Deep Q Learning to solve multi-objective microservice deployment problems. Some work also optimized the request routing decision in the network topology to reduce the path communication delay of services. Casas-Velasco et al. [12] designed a model-less deep reinforcement learning routing algorithm to obtain the optimal route through path state metrics. Qu et al. [15] designed a delay-aware hybrid shortest path heuristic algorithm. However, standalone deployment optimization or routing optimization can only improve processing delay or propagation delay, respectively. In contrast, this paper establishes a joint optimization model for multi-service instance deployment and request routing, utilizing queuing networks for a comprehensive analysis of queuing delay, processing delay, and propagation delay. Additionally, a novel two-stage reward function is designed based on service delay, thereby achieving more efficient and holistic service delay optimization.

2) Joint Service Deployment and Request Routing: Only a small amount of work focused on the joint optimization of service deployment and request routing [16], [18], and it basically provided only a coarse-grained optimization approach. Wang et al. [16] transformed the network into a virtual hierarchical topology and implemented service scheduling using four hierarchical routing algorithms. Wang et al. [18] decoupled the VNF Placement and Routing problem and designed the MADRL-P & R framework using multi-agent deep reinforcement learning. Peng et al. [32] proposed a microservice orchestration algorithm based on solvers, effectively reducing service delay. Hu et al. [33] designed a joint heuristic algorithm to achieve delay optimization in industrial environments. Bi et al. [34] designed a Chebyshev-assisted Actor-Critic deep reinforcement learning algorithm to solve the multi-objective optimization problem of SFC placement routing. Unlike traditional monolithic applications, microservices had flexible splitting and complex data dependencies, which makes fine-grained deployment of microservices more challenging. Therefore, coarse-grained deployment and routing co-optimization did not provide a significant response performance improvement. In our work, we introduce an $M/M/C$ queuing network to meticulously analyze the end-to-end delay composition of services. Building upon this analysis, we optimize the overall energy consumption of the edge scenario, effectively avoiding resource wastage.

C. Energy Optimization

With the rapid growth of network services and edge devices, edge energy consumption is increasing, accounting for 50% of total costs [35]. As existing solutions often neglect energy efficiency, researchers are focusing on service-level energy optimization.

1) Energy-Aware Service Orchestration: Several pieces of work achieved system energy optimization by optimizing resource scheduling. Marotta et al. [21] formulated the SFC placement problem with uncertainty in the required resources and used the robust framework and heuristic algorithms to minimize energy consumption. Varasteh et al. [22] formulated the energy-aware and delay-constrained joint VNF placement and routing as integer linear programming and decomposed it into a two-phase sequential solution of the placement and routing problem. Hazra et al. [23] considered the weighted energy-delay and quality of service constraints for nonlinear planning and proposed a Deep Reinforcement Learning (DRL)-based service deployment framework. Sun et al. [24] proposed an energy-aware routing and adaptive delay shutdown algorithm for Dynamic Service Function Chaining (SFC) deployment to optimize server switching energy consumption. The above work improved resource utilization and reduced the number of resource inputs through rationalized resource scheduling to optimize energy consumption. However, in edge scenarios with differentiated energy efficiencies, pursuing minimizing the number of processors alone does not necessarily reduce the total energy consumption. Therefore, in this paper, we consider the rational allocation of computational resources and the CPU operating frequency of edge servers in an integrated manner to achieve further optimization of energy consumption and service efficiency at the same time.

2) Energy Optimiazation With Dynamic Voltage and Frequency Scaling: DVFS was an effective technique to reduce processor energy consumption [36], [37], but few studies applied DVFS to achieve energy-efficient optimization in real-time services in edge scenarios. Zhou et al. [25] proposed a server energy management solution, EPRONS-Server, to optimize energy consumption for delay-sensitive applications through linear programming with DVFS. Panda et al. [38] proposed an application deadline-aware data offloading solution based on deep reinforcement learning with DVFS. Mo et al. [39] proposed two-step heuristic solutions for the imprecise computation (IC) task mapping problem on multicore platforms

TABLE I
NOTATION AND TERMINOLOGY

Notation	Definition
V, v	the set of edge servers
C_v	the number of CPU cores in edge server v
$\mu_v(f_i, U_v)$	The CPU core processing power in edge server v
F_v, f_i	the set of CPU operating frequencies in edge server v
U_v	the computational resource utilization in edge server v
$e\,(v_i, v_j)$	the communication delay between edge servers v_i and v_j
M, m	the set of microservices
SC, sc	the set of service chains
M_{sc}	the set of microservices contained in a service chain
$L(sc)$	the length of service chain sc
λ_v	the set of average arrival rates of edge servers v
N_v^m	the deployment decision
$P_{v_j,m_b}^{v_i,m_a}$	the routing decision
T^{ave}	the average delay
T_{ene}^{ave}	the average delay after energy consumption optimization
W_b	the basic energy consumption
$W_p(f_i, U_v)$	the processing energy consumption
W_{total}	the total energy consumption of the network

with DVFS, energy, and real-time constraints. However, the above work concentrated on the task offloading domain, and no effort was made on end-to-end delay optimization.

III. MODELS

This section establishes a set of models for static microservice orchestration and dynamic energy conservation. We first introduce the basic concepts and models for underlying edge servers, service chains, microservices, and energy consumption, respectively. We then detail the variables and various constraints in microservice orchestration. Our multi-instance queuing network model enables accurate end-to-end time analysis covering queuing delay, communication delay, and computation delay. This fine-grained model thus facilitates highly efficient microservice orchestration in terms of both delay minimization and energy saving. The notation and terminology used in this paper are provided in Table I.

A. Edge Server Models

We consider a set of edge nodes (i.e., servers) that operate with uninterrupted stability and are free of failures, denoted by $G = (V, E)$, where $v \in V$ signifies edge servers and $e \in E$ denotes links among the edge servers, as illustrated in Fig. 1. The edge server provides the necessary computational resources for deploying microservices instances and processing user requests. In this paper, we denote the computational resource capacity (i.e., the number of CPU cores) of the edge server v as C_v. When a CPU core is shared by multiple microservices, it results in intensive competition for computational resources, potentially causing a decrease in processing power [40]. Hence, we specify that each CPU core can only accommodate a single microservice instance. Additionally, the CPU core processing power $\mu_v(f_i, U_v)$ is also influenced by the CPU operating frequency f_i and the computational resource utilization U_v. We define the set of CPU operating frequencies as $F_v = \{f_1, \cdots, f_{|F_v|}\}$, where $|F_v|$ denotes the number of CPU operating frequencies. Detailed analyses of processing power are discussed in subsection IV-A. Finally, we denote the communication delay between edge servers v_i and v_j as $e(v_i, v_j) \in E$. Notably, the intra-server communication delay is considerably smaller than the inter-server communication delay. Thus, the intra-server communication delay is not considered in this model.

B. Service Deployment Models

In the edge servers, several Internet applications (i.e., services) are deployed in the form of microservices. Each service invokes a series of microservices, structured as a service chain. In this paper, we denote the sets of service chains and microservices as $SC = \{sc_i, \cdots, sc_{|SC|}\}$ and $M = \{m_i, \cdots, m_{|M|}\}$, respectively, where $|SC|$ indicates the number of service chains, and $|M|$ denotes the number of microservices. The microservices contained in the service chain $sc \in SC$ are denoted by the list $M_{sc} = \{m_{sc}^i, \cdots, m_{sc}^{L(sc)}\}$, where $L(sc)$ denotes the length of the service chain $sc \in SC$. Additionally, for precise service deployment, we define the integer variable $N_v^m \in \{0, \cdots, C_v\}$ as the number of microservice instances deployed on the edge server v. However, given that a single server has a finite number of CPU cores, it cannot deploy all microservice instances of the service chain. Hence, the deployment process is subject to the following constraints:

$$\forall v \in V, \sum_{m \in M} N_v^m \leqslant C_v \tag{1}$$

C. Request Routing Models

In edge scenarios, various services continuously receive large-scale user requests and direct them into multiple input queues. We define the set of average request arrival rates on the edge server v as $\lambda_v = \{\lambda_v^i, \cdots, \lambda_v^{|SC|}\}$, where λ_v^i denotes the average request arrival rate of the service chain sc_i on the server v, following the Poisson distribution. Subsequently, services process the user requests by sequentially invoking each microservice in the service chains. To accurately process and route user requests, this model defines the linear variable $P_{v_j,m_b}^{v_i,m_a} \in [0, 1]$ as the probability of forwarding from microservice m_a in edge server v_i to microservice m_j in edge server v_b. Obviously, user requests must be routed to servers where the required microservice instances are deployed. Hence, we impose the following routing constraints:

$$\forall sc \in SC, \quad \sum_{v \in V} P_{v_j,m_b}^{v_i,m_a} = 1 \tag{2}$$

$$\forall sc \in SC, \quad P_{v_j,m_b}^{v_i,m_a} \leqslant \frac{N_v^m}{C_v} \tag{3}$$

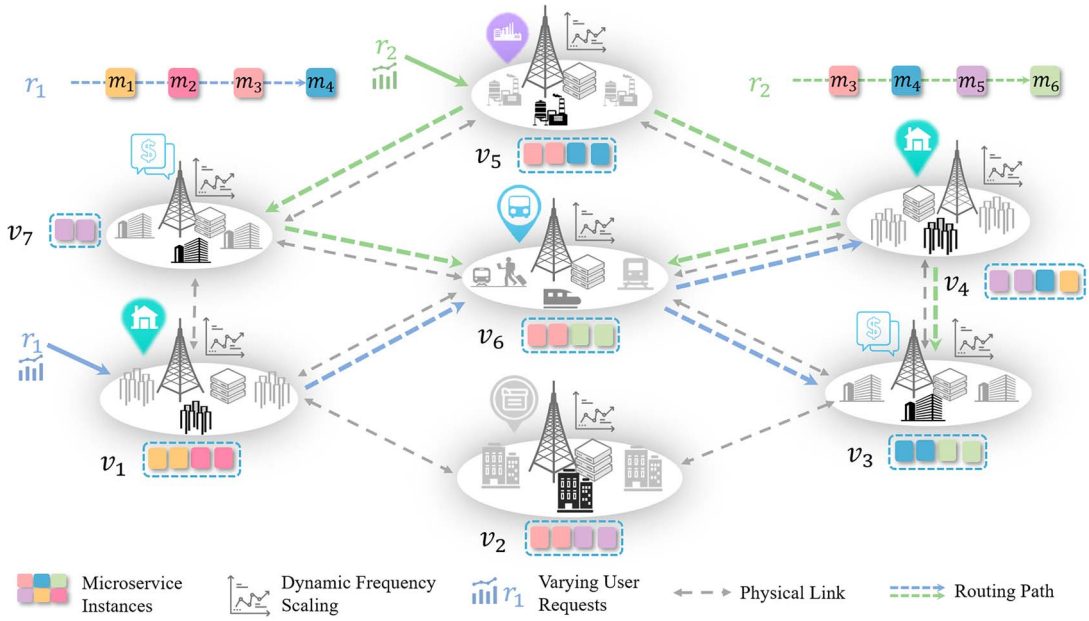


Fig. 1. An example of DVFS-enabled microservice orchestration in edge servers.

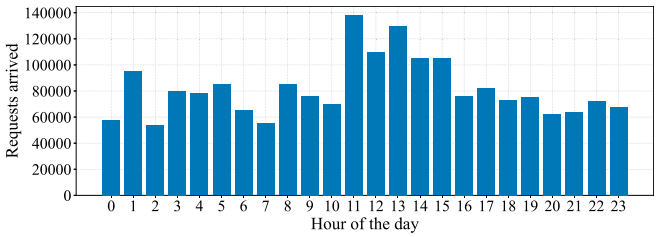


Fig. 2. Varying user request over 24 hours.

When processing user requests, services may share microservice instances. According to the Burke theorem and the linear additivity property of Poisson flows, the average request arrival rate can be aggregated among microservice instances. We define λ_v^m to represent the total request arrival rate at microservice instance m in edge server v, calculated as follows:

$$\forall m \in M,\ \forall v \in V,\ \lambda_v^m = \sum_{sc \in SC} \left(\lambda_{v_i}^{m_i} \times \frac{P_{v,m}^{v_i,m_a}}{\sum_{sc} P_{v,m}^{v_i,m_a}} \right) \quad (4)$$

Meanwhile, the average request arrival rate routed to edge server v should be less than the maximum service capacity of microservice instance m on that server. Thus, we have the following constraint:

$$\forall m \in M,\ \forall v \in V,\ \lambda_v^m \leq N_v^m \mu_v(f_i, U_v) \quad (5)$$

Notably, the service demands in edge scenarios continuously evolve, as shown in Fig. 2. Consequently, different routing decisions are made during peak and off-peak periods.

For clarity, we provide an example. In Fig. 1, we consider two user requests (i.e., service chains) r_1 and r_2, where a series of microservices handle the user requests in the same order as the service chain. Furthermore, Fig. 1 illustrates the topology of a multi-edge network, where there is a certain propagation delay between edge servers. Microservice instances are deployed on the edge servers, and it should be noted that at least one instance of each microservice is deployed in the network. Fig. 1 also depicts the routing path of user request r_1 and how it is assigned to microservice instances. For example, after request r_1 is processed on server v_6, it can be routed to servers v_4 and v_3 for subsequent processing. Therefore, the routing of request r_1 at this point can be $P_{v_4,m_4}^{v_6,m_3} = 1$, or probabilistically $P_{v_4,m_4}^{v_6,m_3} = 0.3$ and $P_{v_3,m_4}^{v_6,m_3} = 0.7$. Through proper deployment of service instances and request routing, we can effectively reduce service delay.

D. Energy Consumption Models

In edge scenarios, where service demand dynamically fluctuates, keeping the CPU at its peak operating frequency for an extended period results in significant energy wastage, as shown in Fig. 2. To enable a fine-grained analysis of energy consumption on the edge, this paper divides the energy consumption of the edge server v into basic energy consumption W_b and processing energy consumption $W_p(f_i, U_v)$ [41]. Specifically, basic energy consumption W_b denotes the energy consumed by the edge server in an idle state, while the processing energy consumption $W_p(f_i, U_v)$ reflects the energy consumed by the edge server when processing user requests. It is worth noting that the processing energy consumption of edge servers, $W_p(f_i, U_v)$, is related to the CPU operating frequency f. Therefore, we can reduce energy consumption while maintaining stable service by adjusting the CPU operating frequency based on varying business demands during different time periods. Detailed analysis and optimization of energy consumption are discussed in Section V.

IV. Joint Microservices Deployment and Request Routing

Since most traditional evolutionary algorithms are based on random exploration strategies, they require a huge iterative cost and are inefficient. In contrast, reinforcement learning (RL) can effectively leverage the agent learning ability to improve exploration efficiency. Additionally, RL can be personalized and optimized according to specific edge environment characteristics. Hence, we have decided to use the RL algorithm to solve the microservice deployment and request routing problem.

This section describes the proposed Delay-Aware Reward Scaling Proximal Policy Optimization (DA-RSPPO) algorithm which derives the delay optimization decisions for static service deployment and routing. Our algorithm designs the reward function as two stages: current action reward and last-step action reward. This innovative design addresses the action space explosion due to the deployment of microservice instances one by one.

A. Delay Optimization Objective

In edge scenarios, the arrival of user requests to the network is probabilistic, with the inter-arrival times of requests being mutually independent. Upon arrival, these requests must be routed to different service instances for processing, and the service delays for individual requests are also independent of one another. The $M/M/C$ model, which characterizes a queueing system where both the arrival and service processes follow Poisson or exponential distributions, with C servers, aligns well with the characteristics of user requests in the edge scenarios. Therefore, referring to existing studies [4], [19], [42], we reasonably assume that the average arrival rate λ of user requests follows a Poisson distribution, modeling the request processing in the service chain as an $M/M/C$ queuing network with the following attributes:

Processing Power: As described in Section III, microservice instances can be deployed on different CPU cores of edge servers. Additionally, the processing power $\mu_{(f_i,U_v)}$ of the CPU cores is strongly correlated with the CPU operating frequency f_i and the server computational resource utilization U_v, in addition to being affected by fixed physical parameters. In this paper, we define the resource utilization U_v of an edge server v as the number of cores occupied by deployed microservice instances:

$$U_v = \frac{\sum_{m \in M} N_v^m}{C_v} \tag{6}$$

According to DVFS, and considering resource utilization U_v, CPU operating frequency f_i, and core base processing power μ_v, we can determine the current core processing power in server v:

$$\mu_v(f_i, U_v) = \mu_v(f_i * U_v + (1 - U_v)) \tag{7}$$

Queuing and Processing Delay: According to the $M/M/C$ network, an average request arrival rate surpassing the processing power would lead to an infinitely long queue, causing a degradation in the network quality of service. In this paper, we precisely define the service intensity of microservice m in edge server v:

$$\forall m \in M, \forall v \in V : \rho_v^m = \frac{\lambda_v^m}{N_v^m \mu_v(f_i, U_v)} < 1 \tag{8}$$

The queuing delay $T_{v,m}^{que}$ for the service in microservice m is a prerequisite for calculating the delay and is computed as follows according to Little law:

$$\forall m \in M, \forall v \in V : T_{v,m}^{que} = \frac{\rho_v^m}{\lambda_v^m (1 - \rho_v^m)^2} p_v^m \tag{9}$$

where these parameters are formulated as follows.

$${\rho_0}_v^m = \frac{\lambda_v^m}{\mu_v(f_i, U_v)} \tag{10}$$

$$p_{0_v}^m = \left[\sum_{i=0}^{N_v^m - 1} \frac{{\rho_0}_v^{m\,i}}{i!} + \frac{N_v^m {\rho_0}_v^{m\,N_v^m}}{N_v^m! (N_v^m - {\rho_0}_v^m)} \right]^{-1} \tag{11}$$

$$p_v^m = \frac{\rho_v^{m N_v^m}}{N_v^m!} p_0^m \tag{12}$$

In summary, the queuing and processing delays $T_{v,m}^{qp}$ can be easily obtained using the convolutions formula of the distribution function, as well as Little law:

$$\forall m \in M, \forall v \in V : T_{v,m}^{qp} = \frac{\rho_v^m}{\lambda_v^m (1 - \rho_v^m)^2} p_v^m + \frac{1}{\mu_v(f_i, U_v)} \tag{13}$$

Communication delay: To precisely analyze the delay, it is essential to consider not only the queuing and processing delay $T_{v,m}^{qp}$ but also the communication delay between instances. We define $\mathbb{P}(sc) = \{p^1, \cdots, p^{|\mathbb{P}(sc)|}\}$ as the routing path set for service chain sc, where $|\mathbb{P}(sc)|$ denotes the number of request routing paths. The routing path follows the order of the microservices contained in M_{sc}. The i-th path $p^i = \{v_{m_1}^{p^i}, \cdots, v_{m_{|M_{sc}|}}^{p^i}\}$ contains the edge servers that it passed through, where $v_{m_j}^{p^i}$ signifies that the request was handled by the microservice m_j in edge server v. The communication delay $T_{p^i}^{com}$ for this path is:

$$T_{p^i}^{com} = \sum_{j=1}^{|p^i|-1} e(v_j^{p^i}, v_{j+1}^{p^i}) \tag{14}$$

Additionally, large-scale user requests (i.e., service chains) adhere to a Poisson distribution. This paper employs an M/M/C queuing model to analyze the long-term performance of service delays across all service chains within edge environments. Therefore, integrating the above discussion with routing decisions $P_{v_j,m_j}^{v_i,m_i}$, the average delay T^{ave} is expressed as follows:

$$T^{ave} = \sum_{j=1}^{|\mathbb{P}(sc)|} \left(\left(\prod_{i=1}^{L(sc)-1} P_{v_j,m_b}^{v_i,m_a} \right) \left(\sum_{n=1}^{|p^i|} T_{v_{m_j}^{p^i}}^{qp} + T_{p^i}^{com} \right) \right) \tag{15}$$

One of the optimization objectives of this paper is to optimize the average service delay through the reasonable deployment of microservice instances as well as routing paths. Based on the

above analysis, the average delay minimization problem in this paper can be formulated as follows:

$$(P1) \quad \min \quad T^{ave} \tag{16}$$

$$s.t. \begin{cases} (1)-(8) \\ N_v^m \in \{0, \cdots, C_v\} \\ P_{v_j,m_b}^{v_i,m_a} \in [0,1] \end{cases} \tag{17}$$

B. Interactive Environment

Deep reinforcement learning enables agents to learn decisions through continuous interaction with the environment. To effectively reduce service delay, we need to map the environment state to reasonable action decisions. Therefore, to obtain historical experience, an interactive environment needs to be designed to actualize the agent learning process.

At each interaction step, the agent experience is represented as a 4-tuple (s_t, a_t, r_t, s_{t+1}). At time t, the agent observes state s_t, takes action a_t, receives reward r_t, and observes new state s_{t+1}. The decision policy $\pi(a_t|s_t)$ maps each state to an action probability, and the optimal policy π^* maximizes the expected cumulative discounted reward $\mathbb{E}_{\pi^*}\left[\sum_t \gamma^t r_t\right]$, where γ is the discount factor. We define three key components of the environment: states, actions, and rewards.

States: The state perceived by the reinforcement learning model mainly includes the normalized distance between the servers of the deployed microservice instances, information about the currently deployed microservice instances, and the remaining computational resources of the servers, which is denoted as:

$$State = \{s | s = (S_E, S_M, S_R)\} \tag{18}$$

where S_E denotes the normalized transmission delay matrix between edge servers, S_M denotes whether the current server has already deployed the same microservice instance, and the remaining computational resources (the number of CPU cores) of each server are denoted by S_R.

Actions: The action of the agent is to deploy microservice instances and route user requests to satisfy model constraints. Specifically, microservice instances can be deployed on edge servers that have different computational resources and communication delays. Hence, when deploying microservice instances, it is necessary to consider the deployment location and the number of instances. However, in real-world scenarios, highly concurrent user requests and complex edge servers lead to large action space, drastically reducing the deployment efficiency of the algorithm. Therefore, we innovatively design the algorithm actions as deploying the required microservice instances one by one, effectively reducing the action space size. The action space of a microservice instance is the edge servers that a microservice instance can select, denoted as follows:

$$Action = \{a_1, a_2, \cdots, a_{|V|}\} \tag{19}$$

where $|V|$ denotes the number of servers in edge scenarios and a_i denotes the action selection probability.

Rewards: Reinforcement learning algorithms based on the Markov Decision Process (MDP) require immediate rewards after each action to evaluate an agent performance. Deploying all microservice instances at once significantly enlarges the action space and could lead to an "action space explosion." It is more practical for the agent to deploy microservice instances one at a time. However, this approach introduces a challenge: the service average delay T^{ave} and corresponding reward can only be calculated after all instances are deployed. This setup prevents accurate evaluation of each action effectiveness before the entire deployment is complete.

To tackle the challenges of computational delays and the sparse nature of reinforcement learning rewards, we develop a novel sparse reward computation function. This function effectively quantifies the merits of each deployment action carried out by the agent while ensuring the efficacy of the training process. Specifically, our objective is to minimize the average delay in the edge scenario with each additional instance of microservices deployed. To achieve this, we compare the average delay of the agent current action with that of the previous action after execution. The portion of delay reduction becomes the reward for the current action. Additionally, in the last step of the action, we consider the average delay reduction in this round compared to the previous round. This is incorporated as part of the reward for the last action, encouraging the agent to prioritize reducing the average delay after all microservice instances are deployed. In summary, the reward function is formulated as follows:

$$r_t = \begin{cases} -\eta_1(T_i^{ave}(t) - T_i^{ave}(t-1)), & t < \tilde{N} \\ -\eta_1(T_i^{ave}(\tilde{N}) - T_i^{ave}(\tilde{N}-1)) - \eta_2(T_i^{ave}(\tilde{N}) - T_{i-1}^{ave}(\tilde{N})), & t = \tilde{N} \end{cases} \tag{20}$$

where $\tilde{N}$ denotes the total number of microservice instances required in the set of service chains SC. $T_i^{ave}(t)$ denotes the average delay after the completion of the t-th action step in the i-th round of training, while $T_i^{ave}(\tilde{N})$ signifies the average delay after the last action step in the i-th round of training, and η_1, η_2 are positive weighting factors.

C. Training Process and Online Decision

As shown in Fig. 3, the DA-RSPPO algorithm comprises an Actor network and a Critic network. The Actor network takes the network state as input and computes the deployment and routing operation to be executed in the current state. After applying the sparse reward shaping function, a reward value is obtained to assess the performance of the behavior. This reward is then utilized as feedback to guide the model in discovering the deployment and routing action that maximizes the objective function. The specific training procedure is outlined in Algorithm 1, with the multi-edge server network G, the set of average arrival rates λ_v for each server, and the hyperparameters as inputs. The outputs include the service deployment and request routing strategy π_θ.

In Algorithm 1, the initialization of the experience storage module Buf, Actor network Λ_θ, and Critic network Δ_θ is performed (Line 1). Subsequently, The algorithm optimizes the deployment of microservices and request routing policies through multiple rounds of training (episodes). Specifically, the

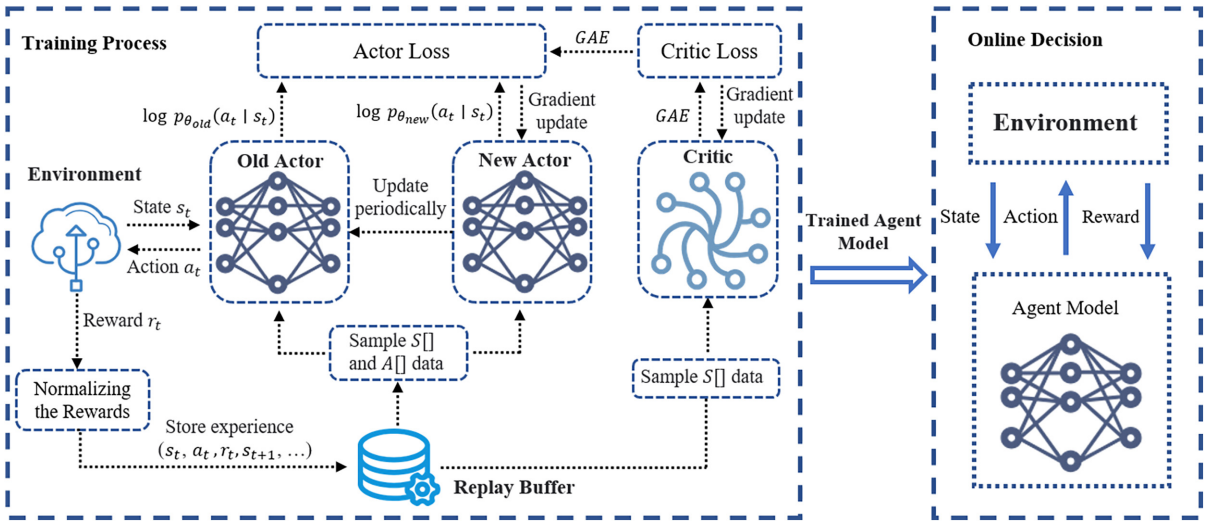


Fig. 3. The architecture of the DA-RSPPO algorithm.

algorithm first calculates the number of required microservice instances based on the average request arrival rate at the server for service chains and performs the initial microservice instance deployment (Lines 3 - 4). The initial deployment solution prioritizes high-demand microservices and deploys instances on servers nearby in hopes of reducing average delay. Then, the algorithm initializes the current state s_t (Line 5) and sets the invalid action mask $mask(a_t)$ in each round of multi-step training (Lines 6 - 8). This setting efficiently excludes invalid actions, thereby enhancing the convergence speed and stability of the algorithm. Further, the action probabilities are sampled using the Actor network to acquire the deployment actions of the microservice instances and compute the routing decision, arrival rate, and average delay respectively (Lines 9). The arrival rate and the average delay are determined by the equations (4) and (15). The routing decision is expressed as follows:

$$P_{v_j,m_b}^{v_i,m_a} = \begin{cases} \alpha\left(\frac{N_v^m}{N_m}\right) + \beta\left[\left(1 - \frac{e(v_i,v_j)}{\sum_{v \in V} e(v_i,v)}\right)/(V(m)-1)\right], & V(m) > 1 \\ N_v^m / N_m, & V(m) = 1 \\ 0, & V(m) = 0 \end{cases} \tag{21}$$

where $P_{v_j,m_b}^{v_i,m_a}$ denotes the routing decision, $V(m)$ denotes the number of servers deploying microservice instances, N_v^m denotes the count of microservice m instances deployed on server v, N_m denotes the total count of currently deployed microservice instances, $e(v_i, v_j)$ denotes the communication delay from server v_i to v_j, and α, β are weighting factors ($\alpha + \beta = 1$).

Based on the above calculations, the reward function is utilized to derive the new state s_{t+1} and the reward value r_t, where the reward value r_t of the action a_t is calculated based on Eq. (20) (Line 10). After obtaining the learning experience $Buf.store(s_t, a_t, r_t, s_{t+1}, ...)$, the algorithm stores it in the experience buffer Buf. When the number of samples in Buf reaches a certain threshold ($Buf.count = update\_batch\_size$), the following steps are executed (Lines 11 - 27):

- Compute the value of the current state and the next state from the Critic network.
- Calculate the Temporal Difference error (TD error) and the Generalized Advantage Estimate (GAE).
- Update the parameters of the Actor and Critic networks to minimize the loss through multiple iterations.

Finally, when the number of training episodes reaches a set value, the algorithm conducts a strategy evaluation. It initializes the state s_t and selects the action with the highest probability as the deployed action a_t in each round of multi-step training to obtain the new state s_{t+1} with the reward value r_t (Lines 28 - 33). If the reward value r_t converges after many iterations, the training process ends, and the service deployment and request routing strategy π_θ is saved (Lines 34 - 40).

V. Energy Optimization With Dynamic Voltage and Frequency Scaling

As service demands dynamically fluctuate (no greater than their peak values), keeping the CPU constantly at its peak operating frequency leads to significant energy wastage. Accordingly, the static orchestration decisions (made by the last section) will be dynamically modified (lowered) by the DVFS-based dynamic energy saving algorithm called EA-DFS. In this way, the sever CPU frequencies are adaptively lowered and the corresponding routing decisions are also modified to match the varying demands and meet the average delay T^{ave}, such that energy consumption is accordingly reduced.

A. Energy Consumption Optimization Objective

In edge scenarios, sustaining a peak CPU operating frequency can effectively enhance request processing efficiency. However, one cannot overlook the significant energy consumption problem associated with edge servers operating at peak frequencies. As outlined in subsection III-D, this paper categorizes the energy consumption of edge scenarios into basic energy consumption and processing energy consumption [41].

Algorithm 1 Delay-Aware Reward Scaling Proximal Policy Optimization Algorithm (DA-RSPPO).

Input:
The edge computing network $G(V, E)$, the set of average arrival rates λ_v for each server, discount factor γ, GAE parameter δ, and clip parameter ε.

Output:
The service deployment and request routing strategy π_θ;

Initialize empirical buffer Buf, Actor network Λ_θ, Critic network Δ_θ;
for $i \in max\_train\_episodes$ **do**
  $\tilde{N}_m \leftarrow \left\lceil \frac{\sum_{v \in V} \lambda_m}{\mu_v(f_i, U_v)} \right\rceil + 1$;
  Deploy microservice instance initial solution;
  Initial state $s_t \leftarrow Env.reset()$;
  for $t \in total\_episode\_steps$ **do**
    $mask(a_t) \leftarrow s_t \lceil |V| : |V| * 2 \rceil$;
    Sample deployment actions from the action probability $a_t \leftarrow sample(probs \leftarrow \Lambda_\theta(s_t, mask(a_t)))$;
    Update routing decision, arrival rate, and average delay;
    Calculate the reward value r_t based on Eq. (20), obtain the new state s_{t+1};
    $Buf.store(s_t, a_t, r_t, s_{t+1}, ...)$
    if $Buf.count == update\_batch\_size$ **then**
      Input $S[s_t, s_{t+1}, ...]$ and $S'[s_{t+1}, s_{t+2}, ...]$ into the Critic network Δ_θ to obtain the values $\Delta_S \leftarrow \Delta_\theta(S)$ and $\Delta_{S'} \leftarrow \Delta_\theta(S')$ for the current and next states;
      $TD \leftarrow r_t + \gamma \Delta'_S - \Delta_S$;
      $GAE \leftarrow \sum_t \gamma^{t-1} \delta^{t-1} \zeta_t$;
      $\Delta_{target} \leftarrow GAE + \Delta_S$;
      for $k \in K\_epochs$ **do**
        Randomly sample learning experiences of size $mini\_batch\_size$ from Buf: $S[...], A[...], R[...], S'[...]$,
        $log\, p_\theta(A[...]|S[...])$;
        Input $S[...]$ into the Actor network Λ_θ to obtain the current action probabilities $p_{\theta'}(A[...]|S[...])$;
        $ratio \leftarrow \frac{p_{\theta'}(A[...]|S[...])}{p_\theta(A[...]|S[...])}$;
        Calculate the Actor network loss $A_{loss} \leftarrow \hat{\mathbb{E}}\lceil min(ratio, clip(r_t, 1-\varepsilon, 1+\varepsilon)) * GAE \rceil$;
        Input $S[...]$ into the Critic network Δ_θ to obtain $\Delta_S \leftarrow \Delta_\theta(S)$;
        Calculate the Actor network loss $C_{loss} \leftarrow mse\_loss(\Delta_{target}, \Delta_S)$;
        Update the Actor and Critic networks using the sampled experiences A_{loss} and C_{loss}, respectively;
      end for
    end if
  end for
  if $i \,\%\, evaluate\_freq == 0$ **then**
    Initial state $s_t \leftarrow Env.reset()$;
    for $t \in total\_episode\_steps$ **do**
      Select the action with the highest probability as the deployment action $a_t \leftarrow Argmax(\Lambda_\theta(s_t, mask(a_t)))$;
      Obtain the new state s_{t+1} and reward value r_t;
    end for
    if reward value r_t convergence **then**
      Save decision $\pi_\theta \leftarrow (\Lambda_\theta, \Delta_\theta)$;
      break;
    end if
  end if
end for
return $\pi_\theta \leftarrow (\Lambda_\theta, \Delta_\theta)$.

Basic Energy Consumption: In edge scenarios, basic energy consumption is primarily influenced by the physical parameters of the edge server. For simplicity, this paper considers basic energy consumption as $85/MJ$ [3].

Processing Energy Consumption: Based on the DVFS theory, processing energy consumption is predominantly determined by the CPU operating frequency f_i and the server resource utilization U_v. Thus, we provide the following definition:

$$W_p(f_i, U_v) = \beta_v U_v f_i^2 \tag{22}$$

where β_v denotes the energy consumption coefficient determined by fixed physical parameters in server v. Then, we can extrapolate the basic energy consumption and processing energy consumption of edge server v to encompass the entire edge scenario. The total edge energy consumption W_{total} is calculated as follows:

$$W_{total} = \sum_{v \in V} (W_p(f_i, U_v) + W_b) \tag{23}$$

It is worth noting that the energy consumption optimization must ensure satisfaction of the service delay while adhering to practical constraints on the current CPU operating frequency of the edge servers. Drawing from the preceding discussion, we can formulate the objective for energy consumption optimization:

$$(P2) \quad \min \quad W_{total} \tag{24}$$

$$s.t. \begin{cases} T_{ene}^{ave} \leqslant T^{ave} \\ \rho_v^m = \frac{\lambda_v^m}{N_v^m \mu_v(f_i, U_v)} < 1 \\ F_v = \{f_1, \cdots, f_{|F_v|}\} \end{cases} \tag{25}$$

B. Energy-Aware Dynamic Frequency Scaling

DA-RSPPO is well-suited for peak hours in edge scenarios, prioritizing low-delay performance during periods of high demand. However, as illustrated in Fig. 2, the type and volume of requests handled by the edge servers fluctuate over time, often including a substantial proportion of low-volume user requests. During such periods, responsive energy-saving strategies should be implemented to mitigate network operation and maintenance costs. Consequently, this subsection introduces the EA-DFS algorithm, dynamically adjusting the CPU operating frequency of each edge server based on the request demand during different time periods. This aims to reduce energy consumption in the network while maintaining a stable quality of service. The pseudo-code and flowchart are shown in Algorithm 2 and Fig. 4, respectively.

In Algorithm 2, First, we get the service deployment and request routing strategy π_θ (Line 1). Then, to reduce the computational difficulty of P2, we relax the integer decision variables $F_v = \{f_1, \cdots, f_{|F_v|}\}$ into linear variables from 0 to 1, and obtain the relaxation solution, i.e., the set of optimal CPU operating frequency F^* for edge servers, by using the Gurobi solver (Line 2). Additionally, by traversing the set F^*, we perform a rounding operation on each element to derive the set of integer CPU operating frequency $\tilde{F}$ for servers (Lines 3-6).

In addition, within the $M/M/C$ queuing network, if the service intensity $\rho_v^m \geqslant 1$, it results in an infinitely long queuing queue. Meanwhile, the CPU operating frequency f_i of an edge server strongly correlates with its CPU processing power $\mu_v(f_i, U_v)$. To uphold service quality and network stability, adjustments to the routing decision in Algorithm 1 are imperative. Initially, we update the CPU processing power $\mu_v(f_i, U_v)^*$ of

Algorithm 2 Energy-Aware Dynamic Frequency Scaling Algorithm (EA-DFS)

Input:
The edge computing network G, the deployment and routing decision of microservice instances π_θ, and the current CPU core processing capability $\mu_v(f_i, U_v)$.

Output:
The set of CPU operating frequencies $\tilde{F}$, routing decision $\tilde{P}$;

1: Get service deployment and request routing strategy π_θ;
2: Solve the relaxation solution in the P2 problem to obtain the set of optimal CPU operating frequencies F^* for the edge servers;
3: $\tilde{F} \leftarrow \emptyset$;
4: **for** $f^* \in F^*$ **do**
5: $\quad \tilde{f}_v \leftarrow round(f^*)$;
6: $\quad \tilde{f}_v$ added to the set $\tilde{F}$;
7: **end for**
8: Update the CPU processing power $\mu_v(f_i, U_v)^*$ in each server according to Eq. (7);
9: $\omega_\mu \leftarrow \frac{\mu_v(f_i, U_v)^*}{\mu_v(f_i, U_v)}$;
10: Update the routing decision in π_θ according to the difference weight ω_μ, Eq. 2, and Eq. 3 to obtain $\tilde{P}$;
11: **return** $(\tilde{F}, \tilde{P})$.

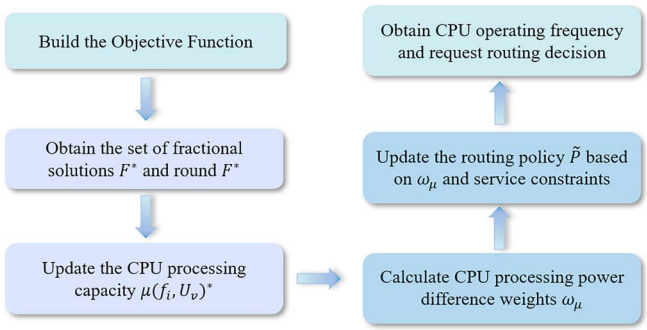


Fig. 4. Flowchart of the EA-DFS algorithm.

each server in accordance with Eq. (7) and compute the deviation weight ω_μ from the initial processing power $\mu_v(f_i, U_v)$ (Lines 7 - 8). Then, the routing decision in π_θ is updated based on the deviation weight ω_μ, Eq. (2), and Eq. (3). Finally, the set of CPU operating frequencies $\tilde{F}$ and the routing decision $\tilde{P}$ are output (Lines 9 - 11).

VI. Performance Evaluation

In this section, we evaluate the overall performance of the methods proposed in this paper through extensive trace-driven simulations. First, our simulation platform consists of two high-performance servers, each equipped with dual i9 CPUs, dual 4090 GPUs, and 128 GB of memory. We also reference the publicly available Alibaba datasets, cluster-trace-microservices-v2021 and cluster-trace-microservices-v2022 [3], which provide runtime metrics of microservices in a production cluster, including call dependencies, response times, call rates, and more. The main metric formats are shown in Table II. Additionally, to facilitate the experiments, we preprocess the datasets by sampling key information such as microservice types, invocation relationships, and required resources from user requests, to construct the foundational data needed for the experiments.

TABLE II
Dataset Format

Notation	Definition
$node$	Node information
$MS\_Resource\_Table$	Microservice runtime
$MS\_Metrics\_Table$	Microservice call rate and response time
$MS\_CallGraph\_Table$	Microservice call graphs

Finally, to ensure reliability and stability, the presented results represent the averages across 200 sets of independent experiments. At the same time, to assess the superiority of the algorithm proposed in this paper, we compare it with three baseline algorithms:

RSDQL [11]: The algorithm employs a deep learning approach for reward sharing based on deep Q-learning. Subsequently, the deployment strategy for microservice instances is derived through an elastic scaling mechanism.

AP [31]: The method constrains the computational resources, adding one to the minimum quantity required for each microservice instance. After multiple iterations, the deployment scheme with the minimum processing delay is obtained.

Holu [22]: The method first deploys service instances using a centrality-based ranking method. Furthermore, the Lagrange Relaxation-based Aggregated Cost algorithm is used to determine the shortest path.

A. Experiment Settings

Physical networks: In this paper, we simulate an edge scenario with a connected service grid consisting of 10 edge servers. In our configuration, each edge server hosts only one server, providing computational resources, i.e., the number of CPU cores, for microservice instances. By default, each edge server is equipped with 20 CPU cores. Consequently, each server can deploy a maximum of 20 microservice instances. Furthermore, each core exhibits heterogeneous processing speeds when deploying various microservice instances.

Service chains and microservices: In previous studies [3], the typical range of microservices varies from 10 to 40, encompassing at least 8 commonly deployed microservice operations. Therefore, in our configuration, aligning with the characteristics of the edge scenario, the default number of service chains is set to 40, with a service chain length ranging from 4 to 6. The services call the microservices contained in the service chains in a specific order to process user requests. It is important to note that the same microservice may coexist within a service chain. Additionally, the average request arrival rate is a key parameter that affects delay and energy consumption. Therefore, we consider the request arrival rate in both peak and off-peak scenarios to validate the effectiveness of the scheme proposed in this paper.

- The peak scenario indicates that a large number of user requests arrive at the edge network within a short period

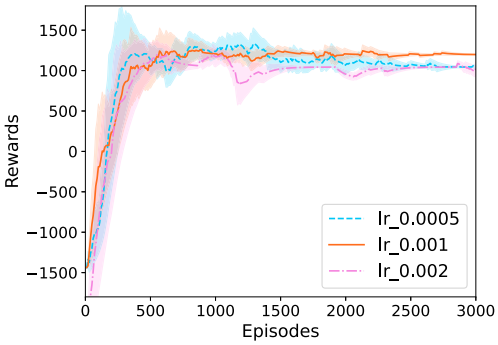


Fig. 5. DA-RSPPO reward curve with different learning rates.

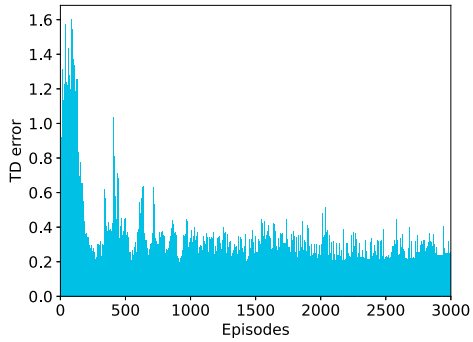


Fig. 6. TD error for DA-RSPPO with a learning rate of 0.001.

of time. Hence, by default, the average request arrival rate for the peak scenario is 10.

- The off-peak scenario indicates that user demand decreases, and the request arrival rate fluctuates within a lower range. Hence, the off-peak scenario is divided into 5 time slots and the average request arrival rate for off-peak scenario is $[4, 8]$.

Experimental indicators: In this paper, we conduct 200 independent experiments, systematically varying key parameters such as the number of edge servers and the number and length of service chains in both peak and off-peak scenarios. To present the results clearly, we consider three metrics: average delay, energy consumption, and deployment cost. Average delay and energy consumption are represented by equations (15) and (23), respectively, while deployment cost is defined as the number of CPU cores used by the microservice instance.

Hyperparameters: To demonstrate the learning process of the DA-RSPPO algorithm, we first examined the impact of different learning rates on the model convergence speed, as shown in Fig. 5. As the learning rate increased from 0.0005 to 0.002, the convergence speed of the algorithm gradually accelerated. However, at a learning rate of 0.0005, the agent final converged normalized reward significantly decreased, indicating that the current learning rate was too low, causing the model to converge to a local optimum. To balance acceleration of convergence with achieving global optimality, we determined that the optimal learning rate for model convergence is 0.001. Further, we present the learning error (TD error) at a learning rate of 0.001. As illustrated in Fig. 6, following multiple learning iterations, the TD error rapidly converges and eventually stabilizes within the range of 0.2 to 0.25, demonstrating the stability and practicality of the DA-RSPPO algorithm. The remaining hyperparameters in the DA-RSPPO algorithm are detailed in Table III.

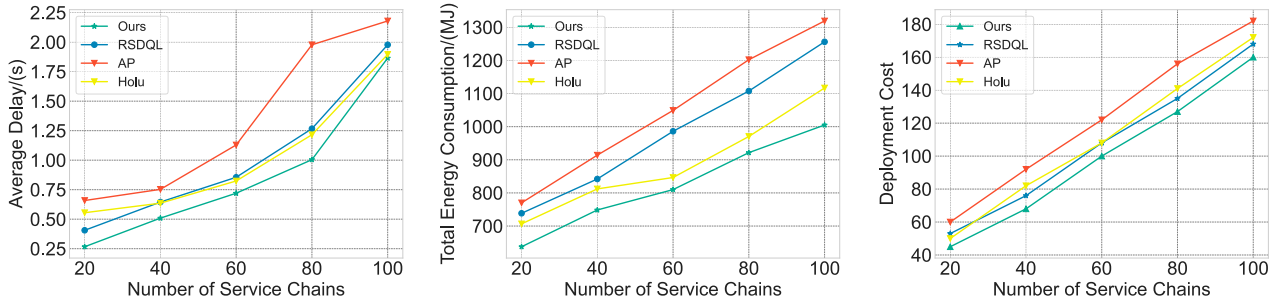
TABLE III
HYPERPARAMETER SETTINGS

Hyperparameters	Value
Learning rate for actor	0.001
Learning rate for critic	0.001
Discount factor	0.9
GAE parameter	0.95
PPO clip parameter	0.2
Optimization rate	(0.9, 0.999)
Adam parameter	0.00001

B. Results

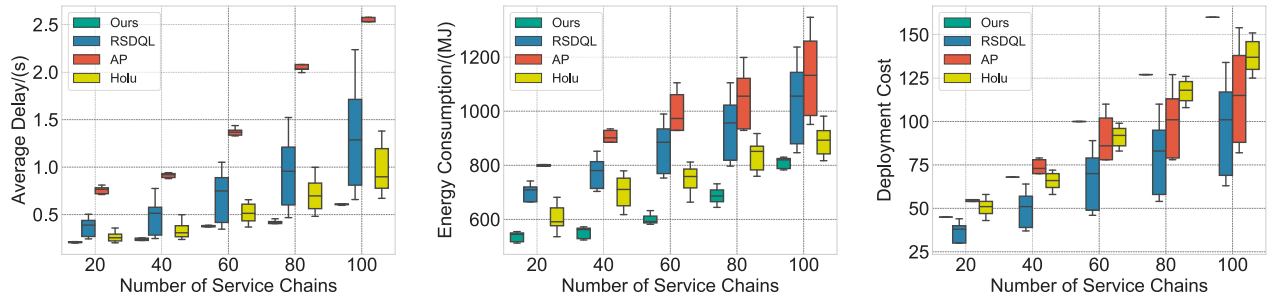
1) The Effect of Number of Service Chains: We first analyze the impact of the number of service chains on average delay, energy consumption, and deployment costs. We consider varying numbers of service chains $[20, 40, 60, 80, 100]$ during peak and off-peak periods, with results presented in Figs. 7 and 8. Fig. 7 shows that, in peak scenarios, as the number of service chains increases, all four algorithms experience rises in average delay, energy use, and deployment costs, but the algorithm proposed in this paper consistently demonstrates superior performance. Additionally, box plots are used to further reveal the performance of the four algorithms in off-peak scenarios, as shown in Fig. 8. The AP, RSDQL, and Holu algorithms exhibit significant fluctuations in average delay and energy consumption, whereas our proposed method shows higher stability. This discrepancy primarily stems from the need for the three baseline algorithms to recalculate the microservice deployment decisions multiple times during off-peak periods, leading to additional delay and energy consumption. In contrast, our algorithm effectively avoids increased delays and energy consumption associated with service redeployment by dynamically adjusting the CPU operating frequency based on existing deployment decisions, such as peak-time deployment decisions. Therefore, even though our deployment costs are slightly higher in off-peak scenarios, our method demonstrates higher efficiency and stability in reducing average delay and energy consumption.

2) The Effect of Length of Service Chain: Subsequently, we investigate the impact of service chain length on experimental outcomes. Utilizing a public dataset from Alibaba, we set service chain lengths to $[[2, 4], [3, 5], [4, 6], [5, 7], [6, 8]]$, with results shown in Figs. 9 and 10. As depicted in Fig. 9(a), when service chain lengths are $[2, 4]$ and $[3, 5]$, our algorithm shows only minor differences compared to RSDQL. However, as the service chain length increases, our algorithm exhibits



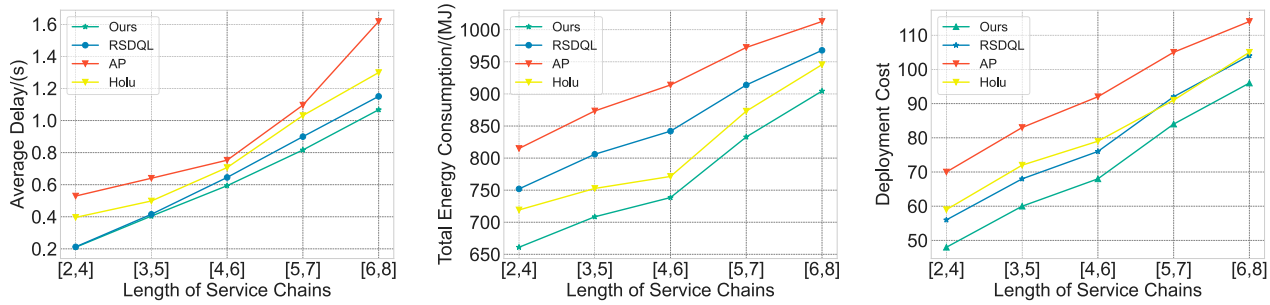
(a) Average delay (b) Total energy consumption (c) Deployment cost

Fig. 7. Results for the number of service chains (peak).



(a) Average delay (b) Total energy consumption (c) Deployment cost

Fig. 8. Results for the number of service chains (off-peak).

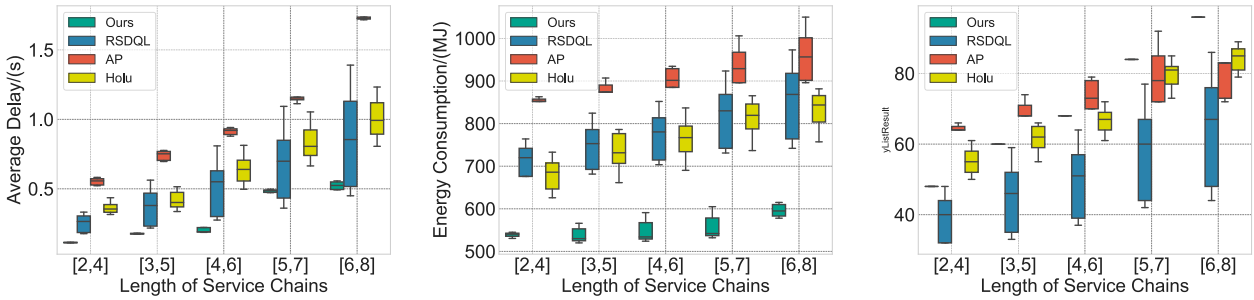


(a) Average delay (b) Total energy consumption (c) Deployment cost

Fig. 9. Results for the service chain length (peak).

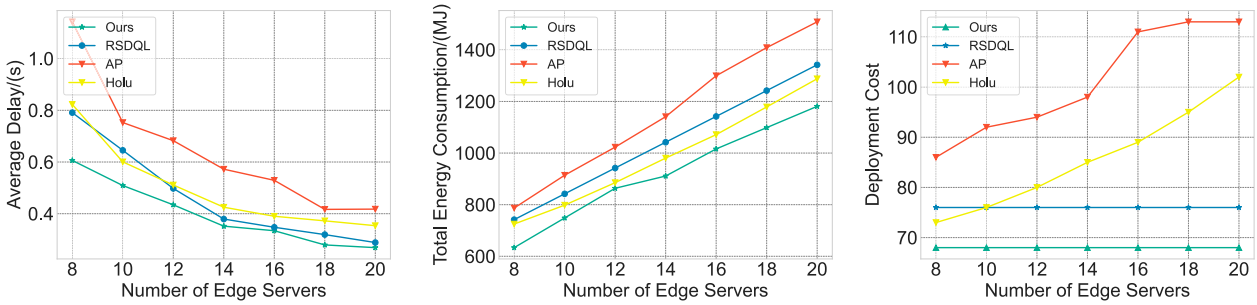
the best performance in terms of average delay. Compared to the three baseline algorithms, it achieves delay improvements of 22.8% (Holu), 32.5% (RSDQL), and 51.6% (AP). This occurs because, through probabilistic routing decisions, our algorithm can effectively reduce queuing and communication delays, offering a more significant advantage in lowering average delay. Moreover, as shown in Fig. 10(c), the deployment cost of our method is higher than that of the other three baseline algorithms and matches the deployment cost under peak scenarios (Fig. 9(c)). This is due to the use of the same deployment strategy during both peak and off-peak conditions, where adjusting the CPU working frequency ensures service quality while optimizing energy consumption. The baseline algorithms, not suitable for dynamic scenarios, require repeated execution of deployment and routing algorithms according to different service demands. While this saves on some deployment costs, it results in additional delays and increased energy consumption.

3) The Effect of Number of Edge Servers: Moreover, we vary the number of edge servers within the range $[8, 10, 12, 14, 16, 18, 20]$ to study their impact on the outcomes. The experimental results are displayed in Figs. 11 and 12. We observe that the average delay of the four algorithms in Figs. 11(a) and 12(a) shows a decreasing trend as the number of edge servers increases. This occurs because as the number of edge servers increases, each server location in the network becomes relatively centralized, reducing communication delays and thus gradually decreasing the average delay. Additionally,



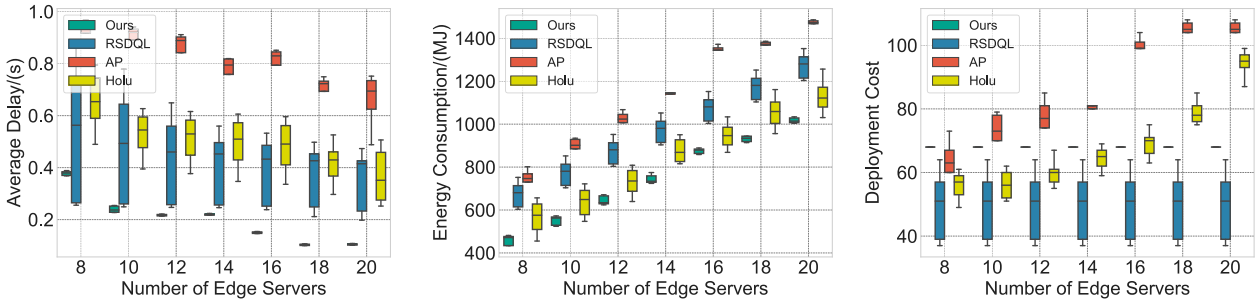
(a) Average delay (b) Total energy consumption (c) Deployment cost

Fig. 10. Results for the service chain length (off-peak).



(a) Average delay (b) Total energy consumption (c) Deployment cost

Fig. 11. Results for the number of edge servers (peak).



(a) Average delay (b) Total energy consumption (c) Deployment cost

Fig. 12. Results for the number of edge servers (off-peak).

as shown in Figs. 11(b) and 12(b), the gradual increase in the number of edge servers provides more deployment location choices for microservice instances, while also increasing the baseline energy consumption of the MEC network. Consequently, the network energy overhead exhibits an upward trend. However, our algorithm still stands out, achieving energy consumption gains of 12.6% (Holu), 14.6% (RSDQL), and 21.8% (AP), as shown in Fig. 11(b). This is because our algorithm effectively reduces the number of active microservice instances needed to process requests through an optimized deployment strategy, thereby optimizing processing energy consumption in the network. Interestingly, the deployment costs for our algorithm and RSDQL remain constant, while those for AP and Holu gradually increase, as shown in Figs. 11(c) and 12(c). This is because the number of instances required by our algorithm and RSDQL depends on the average request arrival rate, independent of the number of servers. In contrast, as the number of edge servers increases, the AP and Holu algorithms deploy more redundant instances, thereby increasing deployment costs.

4) Algorithm Execution Time: Finally, to assess the execution times of various algorithms, we vary the number of service chains and conduct a systematic evaluation of algorithm execution times. As illustrated in Fig. 13, the execution times for all four algorithms demonstrate an upward trend as the number of service chains increases. Notably, when the number of service chains exceeds 80, our method significantly outperforms the RSDQL and Holu algorithms in terms of execution time. This

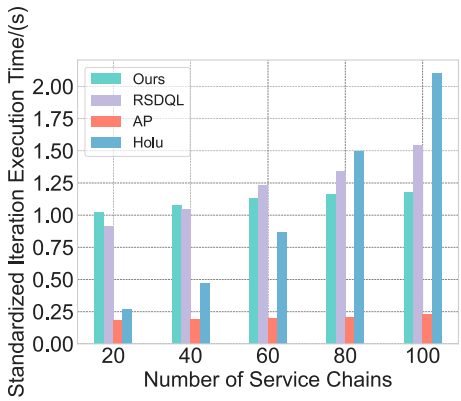


Fig. 13. Algorithm execution time.

performance advantage primarily stems from our innovatively designed two-stage reward function (Equation 20), which effectively reduces the solution complexity of the reinforcement learning algorithm (DA-RSPPO) and accelerates the convergence process, making it more suitable for large-scale service chain scenarios. Additionally, although the AP algorithm, as a heuristic method, shows advantages in execution time, it consistently underperforms in reducing average service delay and network energy consumption compared to other algorithms. Therefore, considering both performance metrics and execution time, our method demonstrates the best overall performance.

C. Scalability Analysis

This paper employs a joint optimization model for microservice deployment and request routing, reducing average delay and network energy consumption. The scalability of the proposed model is reflected in:

Scenario Division: The edge environment is divided into peak and off-peak phases based on service demands. Peak phases focus on optimizing delay, while off-peak phases reduce energy consumption under delay constraints. This design is well-suited for highly variable edge environments, maintaining energy efficiency across scenarios.

Multi-Instance Modeling: Using the $M/M/C$ queuing network, the model supports flexible deployment of different numbers of microservice instances, adapting to various deployment scales, from small applications to large, multi-node systems.

Reward Function and Invalid Action Mask: The RL reward function is designed in two stages, with an added invalid action mask to block ineffective actions. This innovative design effectively mitigates the issue of action space explosion, accelerates algorithm convergence, and is more suitable for large-scale network structures.

VII. Conclusion

This paper has minimized energy and delay by jointly optimizing microservice deployment and request routing through multi-instance modeling, fine-grained orchestration, and dynamic adaptation. We have constructed a set of models based on open Jackson queuing networks. Our multi-instance queuing network model enables accurate end-to-end time analysis, covering the full service process regarding queuing delay, communication delay, and computation delay. Secondly, we have proposed a reinforcement learning approach, referred to as DA-RSPPO. This algorithm has derived the delay optimization decisions for static service deployment and routing. Our innovative design partitions the reward function into two stages: current action reward and last-step action reward, effectively addressing the so-called action space explosion problem. Furthermore, we have leveraged the DVFS technique to design a dynamic energy-saving algorithm which adaptively adjusts server CPU frequencies and request routing decisions, thereby dynamically minimizing energy consumption according to the fluctuating request patterns. Finally, the proposed scheme has been compared with the baseline algorithm using the Alibaba public dataset. Experiment results have demonstrated that our approaches significantly outperform baseline algorithms in both delay and energy consumption.

In the future, we will take service burstiness into account and explore dynamic deployment strategies for microservice instances, along with dynamic routing policies. Additionally, the security and privacy of network services are critical concerns for users. Therefore, developing methods for deploying microservices that safeguard user privacy and ensure service security will also be key issues to address in our future work.

References

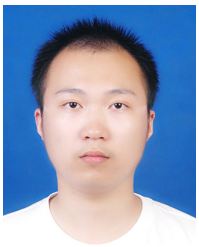
[1] A. Sill, "The design and architecture of microservices," *IEEE Cloud Comput.*, vol. 3, no. 5, pp. 76–80, Sep./Oct. 2016.
[2] "DAPR," Microsoft. Accessed: Mar. 20, 2024. [Online]. Available: https://dapr.io/
[3] "Alibab cluster data," Alibaba. Accessed: Mar. 20, 2024. [Online]. Available: https://github.com/alibaba/clusterdata
[4] K. Peng et al., "Delay-aware optimization of fine-grained microservice deployment and routing in edge via reinforcement learning," *IEEE Trans. Netw. Sci. Eng.*, vol. 11, no. 6, pp. 6024–6037, Nov./Dec. 2024.
[5] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys & Tut.*, vol. 19, no. 4, pp. 2322–2358, Aug. 2017.
[6] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys & Tut.*, vol. 19, no. 3, pp. 1628–1656, Mar. 2017.
[7] S. Guo, Y. Qi, Y. Jin, W. Li, X. Qiu, and L. Meng, "Endogenous trusted DRL-based service function chain orchestration for IoT," *IEEE Trans. Comput.*, vol. 71, no. 2, pp. 397–406, Feb. 2022.
[8] F. Guan, J. Qiao, and Y. Han, "DAG-fluid: A real-time scheduling algorithm for DAGs," *IEEE Trans. Comput.*, vol. 70, no. 3, pp. 471–482, Mar. 2021.
[9] F. Guan, L. Peng, and J. Qiao, "A fluid scheduling algorithm for DAG tasks with constrained or arbitrary deadlines," *IEEE Trans. Comput.*, vol. 71, no. 8, pp. 1860–1873, Aug. 2022.
[10] J. Luo, J. Li, L. Jiao, and J. Cai, "On the effective parallelization and near-optimal deployment of service function chains," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1238–1255, May 2021.
[11] W. Lv et al., "Microservice deployment in edge computing based on deep Q learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2968–2978, Nov. 2022.
[12] D. M. Casas-Velasco, O. M. C. Rendon, and N. L. S. da Fonseca, "DRSIR: A deep reinforcement learning approach for routing in software-defined networking," *IEEE Trans. Netw. Service Manage.*, vol. 19, no. 4, pp. 4807–4820, Dec. 2022.
[13] B. Li et al., "READ: Robustness-oriented edge application deployment in edge computing environment," *IEEE Trans. Services Comput.*, vol. 15, no. 3, pp. 1746–1759, May/Jun. 2022.

[14] B. Németh, N. Molner, J. Martín-Pérez, C. J. Bernardos, A. de la Oliva, and B. Sonkoly, "Delay and reliability-constrained VNF placement on mobile and volatile 5G infrastructure," *IEEE Trans. Mobile Comput.*, vol. 21, no. 9, pp. 3150–3162, Sep. 2022.
[15] L. Qu, C. Assi, M. J. Khabbaz, and Y. Ye, "Reliability-aware service function chaining with function decomposition and multipath routing," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 2, pp. 835–848, Jun. 2020.
[16] Y. Wang, C.-K. Huang, S.-H. Shen, and G.-M. Chiu, "Adaptive placement and routing for service function chains with service deadlines," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 3, pp. 3021–3036, Sep. 2021.
[17] S. Yang, F. Li, S. Trajanovski, X. Chen, Y. Wang, and X. Fu, "Delay-aware virtual network function placement and routing in edge clouds," *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 445–459, Feb. 2021.
[18] S. Wang, C. Yuen, W. Ni, Y. L. Guan, and T. Lv, "Multiagent deep reinforcement learning for cost- and delay-sensitive virtual network function placement and routing," *IEEE Trans. Commun.*, vol. 70, no. 8, pp. 5208–5224, Aug. 2022.
[19] Y. Hu, H. Wang, L. Wang, M. Hu, K. Peng, and B. Veeravalli, "Joint deployment and request routing for microservice call graphs in data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 11, pp. 2994–3011, Nov. 2023.
[20] B. Xu, J. Guo, F. Ma, M. Hu, W. Liu, and K. Peng, "On the joint design of microservice deployment and routing in cloud data centers," *J. Grid Comput.*, vol. 22, p. 42, Mar. 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:268743782
[21] A. Marotta, F. D'Andreagiovanni, A. Kassler, and E. Zola, "On the energy cost of robustness for green virtual network function placement in 5g virtualized infrastructures," *Comput. Netw.*, vol. 125, pp. 64–75, Oct. 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128617301767
[22] A. Varasteh, B. Madiwalar, A. Van Bemten, W. Kellerer, and C. Mas-Machuca, "Holu: Power-aware and delay-constrained vnf placement and chaining," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 2, pp. 1524–1539, Jun. 2021.
[23] A. Hazra, M. Adhikari, T. Amgoth, and S. N. Srirama, "Intelligent service deployment policy for next-generation industrial edge networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 9, no. 5, pp. 3057–3066, Sep./Oct. 2022.
[24] G. Sun, R. Zhou, J. Sun, H. Yu, and A. V. Vasilakos, "Energy-efficient provisioning for service function chains to support delay-sensitive applications in network function virtualization," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6116–6131, Jul. 2020.
[25] L. Zhou, C.-H. Chou, L. N. Bhuyan, K. K. Ramakrishnan, and D. Wong, "Joint server and network energy saving in data centers for latency-sensitive applications," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, 2018, pp. 700–709.
[26] Y. Xiao, Y. Xue, S. Nazarian, and P. Bogdan, "A load balancing inspired optimization framework for exascale multicore systems: A complex networks approach," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)*, 2017, pp. 217–224.
[27] G. Ma, Y. Xiao, T. Willke, N. Ahmed, S. Nazarian, and P. Bogdan, "A distributed graph-theoretic framework for automatic parallelization in multi-core systems," *Proc. Mach. Learn. Syst.*, vol. 3, pp. 550–568, Apr. 2021.
[28] D. Petrocelli, A. De Giusti, and M. Naiouf, "Collaborative, distributed, scalable and low-cost platform based on microservices, containers, mobile devices and cloud services to solve compute-intensive tasks," in *Proc. Eur. Conf. Parallel Process.*, Cham, Switzerland: Springer, 2021, pp. 545–548.
[29] J. Gedeon, M. Wagner, J. Heuschkel, L. Wang, and M. Muhlhauser, "A microservice store for efficient edge offloading," in *Proc. IEEE Global Commun. Conf. (GLOBECOM),* Piscataway, NJ, USA: IEEE Press, 2019, pp. 1–6.
[30] R. Mahapatra, B. H. Ahn, S.-T. Wang, H. Xu, and H. Esmaeilzadeh, "Exploring efficient ML-based scheduler for microservices in heterogenous clusters," in *Proc. Mach. Learn. Comput. Archit. Syst.*, 2022, pp. 1–6.
[31] T. Z. J. Fu, J. Ding, R. T. B. Ma, M. Winslett, Y. Yang, and Z. Zhang, "DRS: Auto-scaling for real-time stream analytics," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3338–3352, Dec. 2017.
[32] K. Peng, L. Wang, J. He, C. Cai, and M. Hu, "Joint optimization of service deployment and request routing for microservices in mobile edge computing," *IEEE Trans. Services Comput.*, vol. 17, no. 3, pp. 1016–1028, May/Jun. 2024.
[33] M. Hu et al., "Collaborative deployment and routing of industrial microservices in smart factories," *IEEE Trans. Ind. Inform.*, vol. 20, no. 11, pp. 12758–12770, Nov. 2024.
[34] Y. Bi, C. C. Meixner, M. Bunyakitanon, X. Vasilakos, R. Nejabati, and D. Simeonidou, "Multi-objective deep reinforcement learning assisted service function chains placement," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 4, pp. 4134–4150, Dec. 2021.
[35] "What is big data? Bringing big data to the enterprise." 2017. Accessed: Mar. 20, 2024. [Online]. Available: http://www.ibm.com/big-data/us/en/
[36] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron, "PowerPack: Energy profiling and analysis of high-performance systems and applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, no. 5, pp. 658–671, May 2010.
[37] Z. Zhang, Y. Zhao, H. Li, C. Lin, and J. Liu, "DVFO: Learning-based DVFS for energy-efficient edge-cloud collaborative inference," *IEEE Trans. Mobile Comput.*, vol. 23, no. 10, pp. 9042–9059, Oct. 2024.
[38] S. K. Panda, M. Lin, and T. Zhou, "Energy-efficient computation offloading with DVFS using deep reinforcement learning for time-critical IoT applications in edge computing," *IEEE Internet Things J.*, vol. 10, no. 8, pp. 6611–6621, Apr. 2023.
[39] L. Mo, A. Kritikakou, and O. Sentieys, "Energy-quality-time optimized task mapping on DVFS-enabled multicores," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2428–2439, Nov. 2018.
[40] B. Bermejo, C. Juiz, and C. Guerrero, "On the linearity of performance and energy at virtual machine consolidation: The CiS2 index for CPU workload in server saturation," in *Proc. IEEE 20th Int. Conf. High Perform. Comput. Commun.; IEEE 16th Int. Conf. Smart City; IEEE 4th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, 2018, pp. 928–933.
[41] P. Jin, X. Hao, X. Wang, and L. Yue, "Energy-efficient task scheduling for CPU-intensive streaming jobs on hadoop," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 6, pp. 1298–1311, Jun. 2019.
[42] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi, "Multiobjective optimization for computation offloading in fog computing," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 283–294, Feb. 2018.

Liangyuan Wang (Graduate Student Member, IEEE) is currently working toward the Eng.D. degree with the School of Electronic Information and Communications, Huazhong University of Science and Technology. His research interests include edge computing and distributed systems.

Xudong Liu is currently working toward the M.E. degree with the School of Electronic Information and Communications, Huazhong University of Science and Technology. His research interests include LLMs, reinforcement learning, and edge computing.

Haonan Ding is currently working toward the M.E. degree with the School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan, China. His research interests include edge computing and microservice application.



Yi Hu (Graduate Student Member, IEEE) is currently working toward the Ph.D. degree with the School of Electronic Information and Communications, Huazhong University of Science and Technology. His research interests include cloud computing and distributed systems.

Kai Peng received the B.E., M.S., and Ph.D. degrees from Huazhong University of Science and Technology, in 1999, 2002, and 2006, respectively. He is a Professor with the School of Electronic Information and Communications, Huazhong University of Science and Technology, China. His research interests include cloud computing and computer networks.

Menglan Hu received the B.E. degree in electronic and information engineering from Huazhong University of Science and Technology, China, in 2007, and the Ph.D. degree in electrical and computer engineering from the National University of Singapore, Singapore, in 2012. He is an Associate Professor with the School of Electronic Information and Communications, Huazhong University of Science and Technology, China. His research interests include cloud computing, computer networks, distributed systems, and mobile computing.