World Wide Web (2022) 25:1769–1792
https://doi.org/10.1007/s11280-021-00939-7

DIMA: Distributed cooperative microservice caching for internet of things in edge computing by deep reinforcement learning

Hao Tian[1] · Xiaolong Xu[1,2,3,4,5] · Tingyu Lin[6] · Yong Cheng[7] · Cheng Qian[8] · Lei Ren[9] · Muhammad Bilal[10]

Received: 14 July 2021 / Revised: 5 August 2021 / Accepted: 16 August 2021 /
Published online: 24 August 2021

Abstract
The ubiquitous Internet of Things (IoTs) devices spawn growing mobile services of applications with computationally-intensive and latency-sensitive features, which increases the data traffic sharply. Driven by container technology, microservice is emerged with flexibility and scalability by decomposing one service into several independent lightweight parts. To improve the quality of service (QoS) and alleviate the burden of the core network, caching microservices at the edge of networks empowered by the mobile edge computing (MEC) paradigm is envisioned as a promising approach. However, considering the stochastic retrieval requests of IoT devices and time-varying network topology, it brings challenges for IoT devices to decide the caching node selection and microservice replacement independently without complete information of dynamic environments. In light of this, a MEC-enabled distributed cooperative microservice caching scheme, named DIMA, is proposed in this paper. Specifically, the microservice caching problem is modeled as a Markov decision process (MDP) to optimize the fetching delay and hit ratio. Moreover, a distributed double dueling deep Q-network (D3QN) based algorithm is proposed, by integrating double DQN and dueling DQN, to solve the formulated MDP, where each IoT device performs actions independently in a decentralized mode. Finally, extensive experimental results are demonstrated that the DIMA is well-performed and more effective than existing baseline schemes.

Keywords Internet of things · Mobile edge computing · Microservice · Edge caching · Deep reinforcement learning

This article belongs to the Topical Collection: *Special Issue on Resource Management at the Edge for Future Web, Mobile and IoT Applications*
Guest Editors: Qiang He, Fang Dong, Chenshu Wu, and Yun Yang

✉ Xiaolong Xu
xlxu@ieee.org

Extended author information available on the last page of the article.

1 Introduction

Nowadays, the unprecedented advancement of Internet of Things (IoTs) is flourishing a huge potential for provisioning ubiquitous services of the mobile applications (e.g., augmented reality, mobile gaming, and image recognition) in the fifth-generation (5G) mobile networks [13, 22, 28]. These services are not only hungry for numerous computational resources, but also endure the rigorous real-time requirements to support the mobile applications [1, 29, 44]. With the proliferation of IoT devices, the amount of data traffic raises extremely. According to the report [14], it is estimated that by 2023 almost 110 exabytes of data traffic are produced from IoT devices worldwide every month. However, due to the constrained computation and storage capabilities as well as the limited battery lifetime imposed on IoT devices, it is challenging to processing delay-sensitive and computationally-intensive services. Furthermore, the explosive growth of data traffic increases the communication burden of the core network.

In light of this, mobile edge computing (MEC) is emerging as a promising technology to meet the low-latency demands [18, 19, 34]. With MEC, edge nodes (ENs) are not only employed with the computing and storing resources, but also deployed kinds of microservices in proximity to IoT devices at the edge of networks [7, 24, 38]. Particularly, driven by the container technology (e.g., Docker), the services can be decomposed as several parts of microservices in which each one can be packaged as one container image to execute independently, which improves the flexibility and scalability greatly [10, 30]. As the perspective of service providers, e.g., YouTube, Facebook, and Weibo, deploying and caching microservices at ENs is a promising approach to enable IoT devices to retrieve desire microservices at the network edge with ultra-low transmission time instead of cloud data centers [41, 42]. The fetching latency of IoT devices can be cut down if the target microservice is cached on nearby ENs. With the microservice caching at ENs, the data traffic between IoT devices and core network is mitigated shapely, thereby the network congestion is also alleviated [4].

Although caching microservices at the edge of networks reduces the response delay, it also brings challenges to achieve high caching utilization and efficiency by considering the highly distributed and time-varying retrieval requests from IoT devices in MEC. First, due to the dynamic topology of networks and mobility of IoT devices, which candidate is an appropriate EN for IoT devices to meet the retrieval requirements and improve the quality of service (QoS) if the IoT device's local caching buffer misses the request. Second, the storage capacity of each ENs is too limited to cache all the microservices. If both the local buffer and the target EN miss the requested microservice, the IoT device fetches the desired one from the cloud through the backhaul link, which increases the transmission time and degrades the QoS. Intuitively, consider that each EN is equipped with the caching and communication capabilities, ENs should not operate the retrieval requests independently. Therefore, it is essential for ENs to cooperate and share with each other to improve the hit ratio and caching efficiency. Furthermore, when the caching list is full in ENs, the microservice replacement is needed. However, existing replacement strategies, e.g., least recently used (LRU) [21], are not proper for the dynamic environments.

Thus, in this paper, we propose a deep reinforcement learning (DRL) based distributed cooperative microservice caching scheme, named DIMA. Recently, DRL, as a nature-inspired approach, is empowered the cognitive capabilities in dynamic environments, which can reduce the complexity of solutions compared with the conventional approaches (e.g., convex optimization) [5, 11, 31, 36, 45]. Therefore, the DRL-based approach is utilized to address the microservice caching problems. First of all, the proposed cooperative microservice caching problem is modeled as a Markov decision process (MDP) to improve the

retrieval delay with the cooperation of ENs. Then, to solve the formulated problem, we integrate two existing DRL techniques, i.e., double deep Q-network (DQN) [6] and dueling DQN [32], to propose a double dueling DQN (D3QN) based algorithm. Finally, we conduct extensive simulations to evaluate the performance of DIMA in terms of the fetching delay and hit ratio. The main contributions in this paper are as follows.

- Model the microservice caching problems, i.e., caching mode selection and microservice replacement, as a MDP to minimize the fetching latency taking into account the cooperation of ENs.
- Propose a D3QN-based distributed algorithm to solve the formulated optimization issues without any prior knowledge, which enables each IoT device performs action independently.
- Conduct extensive simulations to evaluate the performance of DIMA, and numerical results are demonstrated that DIMA is well-performed compared to the existing baseline approaches.

The organization of this paper is as follows. In Section 2, related work is reviewed regarding MEC and IoT systems, edge caching, and DRL for edge caching, respectively. Section 3 introduces the system model of MEC-enabled microservice caching. In Section 4, the proposed model is formulated as a Markov decision process. Section 5 presents the details of the DIMA. Section 6 discusses the experimental results. Finally, in Section 7, the conclusion is drawn.

2 Related work

2.1 Mobile edge computing and IoT systems

Recently, MEC is envisioned as a promising paradigm with the ultra-low delay, where the ENs are closer to IoT devices. With the computing and storage capabilities, ENs enable multiple processing requirements of applications from IoT devices, which reduces the completion latency and improves the QoS [15, 33, 37]. In [25], Shi et al. reviewed the prospect and challenges of MEC with the ubiquitous IoT devices. MEC pushes the computing resources to the edge of networks, which has the potential to release the data processing and battery life burden of IoT devices. To decrease the latency and power cost of resource-constrained IoT devices, Lei et al. [12] designed a collaborative computation offloading and scheduling algorithm by utilizing NB-IoT technology. The problem was formulated as a CTMDP, which is solved by the approximate dynamic programming. The results demonstrated that the proposed algorithm can handle the multi-scale IoT devices and reduce the time and energy overhead significantly. In [9], Hu et al. investigated the joint computation offloading and scheduling issues by considering the finite processing capabilities and stochastic offloading requests. An i-NSGA-II based multi-objective algorithm is advocated to solve the formulated optimization problem. Furthermore, consider that the ENs and IoT devices are both with the computing capabilities, the synergy between ENs and IoT devices can enable the potential of cooperative computing and low processing delay. In [8], Hong et al. gave a distributed game theory-based algorithm to solve the multi-hop computation offloading issues to face the challenges of increasing requirements of resource-hungry IoT applications.

2.2 Edge caching

Nowadays, with the explosion of pervasive mobile applications, the ENs are not only with the computing resources, but also with the caching capabilities. Such that the data can be stored in ENs for retrieval at the edge of networks, which cuts down the fetching delay from the core network and transmission of backhaul link, and improves the QoS of IoT devices. To alleviate the burden of caching storage, Xie et al. [35] considered a probabilistic edge caching scheme to achieve the balance of caching cost and hit ratio. By utilizing a two-layer searching algorithm, the proposed nonconvex optimization problem to improve the storage efficiency is solved. Intuitively, edge caching is greatly implemented for the computation offloading, which mitigates the mobile data traffic. In [2], Bi et al. investigated a joint computation offloading and edge caching problem and formulated such a problem as mixed-integer non-linear programming. To reduce the completion latency and energy cost, a series of optimization algorithms are presented. Moreover, IoT devices are the potential caching node to serve the retrieval requests from neighboring devices. In [40], Zhang et al. proposed a device-to-device (D2D) based content caching problem with the multi-winner auction to optimize the caching efficiency. Then a semidefinite programming technique is adopted to achieve the optimal solution. As a result, the designed algorithms effectively optimize the data traffic load and retrieval latency.

2.3 Deep reinforcement learning for edge caching

Due to the stochastic caching requests and dynamic network communications, some conventional methods (e.g., convex optimization) are hardly achieve the optimal solutions in the time-varying environment [26]. DRL is a promising technology to enable the agent to interact with the environment to explore the optimal action [3, 16, 23]. In [20], Wu et al. proposed a DRL-based algorithm to optimize the cache hit ratio with the stochastic content popularity. The algorithm enables the ENs to update the caching list under the dynamic retrievals without any prior knowledge of the content popularity. To improve the caching efficiency in the time-varying file popularity, Zhong et al. [43] designed a centralized and decentralized caching algorithm based on the actor-critic DRL, respectively. The computation offloading and edge caching can also be optimized collaboratively by utilizing DRL technology. In [39], Zhang et al. investigated the social-aware MEC envisioned edge computing and caching issues. Considering the limited resources on ENs and the high mobility of vehicles, a DRL-based scheme is given to optimize the scheduling of content with the goal of utility maximization.

To the best of our knowledge, only a few works investigate the microservice caching problem in the MEC-empowered IoT systems, where each IoT device can perform the caching decisions in a decentralized manner. In view of this, we propose a distributed cooperative microservice caching scheme based on the D3QN in this paper for each IoT device to achieve the optimal caching strategies independently with no prior knowledge.

3 System model

In this section, the system model of the cooperative microservice caching in the MEC-envisioned IoT networks is constructed. Specifically, the system model includes the network model, request model, wireless communication model, and delay model.

3.1 Network model

As illustrated in Figure 1, we consider a 3-tier collaborative network system over 5G connection in this paper. There are one centralized cloud data center, M ENs, N IoT devices, and S microservices. Denote $\mathcal{M} = \{1, 2, \ldots, M\}$ as the set of ENs and denote $\mathcal{N} = \{1, 2, \ldots, N\}$ as the set of IoT devices. Furthermore, the set of microservices is defined as $\mathcal{S} = \{1, 2, \ldots, S\}$. Without loss of generality, the system model is running in the discrete time slot mode. We use $\mathcal{T} = \{1, 2, \ldots, T\}$ to denote the set of time slots. Consider the sufficient storing capacities, it is assumptively that all the microservices in the $\mathcal{S}$ are deployed at the cloud data centers. Therefore, IoT devices can fetch all the S microservices from the remote cloud. On contrast, both the ENs and IoT devices have the limited storage capabilities, which only cache the part of microservices. Denote $c_n^{UE}(\forall n \in \mathcal{N})$ and $c_m^{EN}(\forall m \in \mathcal{M})$ as the caching capacity of IoT devices n and EN m, respectively. Assume that each microservice is independent and cannot be partitioned as the microservices. Through the 5G wireless links, each IoT device connects to the ENs and centralized cloud to retrieve the requested microservice.

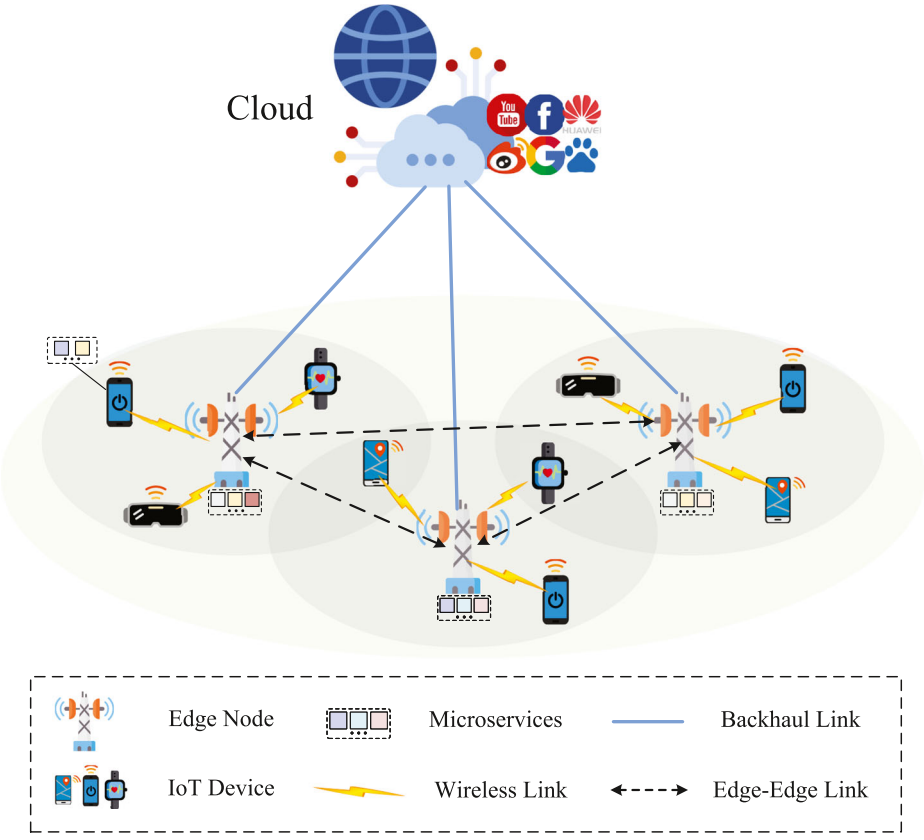

Figure 1 Architecture of MEC-envisioned cooperative microservice caching

3.2 Microservice request model

In the constructed system, assume that one or zero microservice fetching requests of one IoT device generated by users at each time slot t. For the microservice $s \in S$, let z_s denote the size of s, and then the size of microservice vector is defined as $Z = [z_1, z_2, \ldots, z_S]$. The microservice request variable is denoted by $r_n^s(t)$, which is given by

$$r_n^s(t) = \begin{cases} 1, & \text{if IoT device } n \text{ requests microservice } s \text{ at time slot } t, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

The microservice request vector of IoT device n is defined as $\mathcal{R}_n(t) = [r_n^1(t), r_n^2(t), \ldots, r_n^S(t)]$. Therefore, the microservice request vector of all the IoT devices at time slot t is defined as $\mathcal{R}(t) = [\mathcal{R}_1(t), \mathcal{R}_2(t), \ldots, \mathcal{R}_N(t)]$. Note that the microservice requests are randomly generated from users in the stochastic MEC-envisioned IoT systems, we assume that the microservice request follows the independent and identical distribution.

Due to the ENs deployed closer to IoT devices, the microservices retrieval time is less than fetching from the centralized cloud, if the local storage does not cache it. The associate variable between IoT device n and EN m is presented as

$$a_{m,n}^s(t) = \begin{cases} 1, & \text{if IoT device } n \text{ requests to EN } m \text{ for microservice } s, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The microservice replacement occurs when the caching list is full. Let $\varphi_n(t) \in \mathcal{S}$ represent the replacement microservice decision for IoT device n. As a result, when the caching list is full and the request from IoT device n is missed, the microservice $\varphi_n(t)$ is replaced with microservice s in time slot t.

3.3 Wireless communication model

Supposing that each EN has the same communication range to serve the microservice fetching requests. The wireless communication links between IoT devices and ENs, as well as ENs and ENs, are based on orthogonal frequency multiple-access (OFDMA) technology. Such that the communication nodes are not inferenced with each other. Let $\lambda_n^m(t)$ denote the binary variable to represent the association between IoT device n and EN m at time slot t, where $\lambda_n^m(t)$ is given by

$$\lambda_n^m(t) = \begin{cases} 1, & \text{if IoT device } n \text{ access to EN } m, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Note that, at each time slot t, there is only one EN be selected for IoT device n. The signal to noise ratio (SNR) between IoT device n and EN m is calculated as

$$\xi_{m,n}(t) = \frac{p_{m,n}(t) h_{m,n}(t)}{\sigma^2}, \tag{4}$$

where $p_{m,n}(t)$ is the transmission power between IoT device n and EN m, $h_{m,n}(t)$ is the channel gain between IoT device n and EN m, and σ^2 is the Gauss white noise power. Thus, the transmission rate between IoT device n and EN m in time slot t is represented by

$$\beta_{m,n}(t) = \lambda_n^m(t) B_{m,n} \log_2 \left(1 + \xi_{m,n}(t)\right), \tag{5}$$

where $B_{m,n}$ is the bandwidth between IoT device n and EN m. To improve the QoS of users, the cooperation among ENs to cache microservices enables the retrieval time to decrease. Otherwise, if one EN misses the request, it has to fetch the target microservice

from the remote cloud, which needs more transmission latency. Similarly, the transmission rate between EN m_1 and m_2 in time slot t is given by

$$\beta_{m1,m2}(t) = B_{m1,m2} \log_2 \left(1 + \xi_{m1,m2}(t)\right). \tag{6}$$

3.4 Delay model

In this paper, there are four ways for IoT devices to fetch the requested microservices. Let $x_n^s(t)$ and $x_m^s(t)$ denote the caching variable of IoT device n and EN m, where $x_n^s(t)$ and $x_m^s(t)$ are given by

$$x_n^s(t) = \begin{cases} 1, & \text{if microservice } s \text{ is cached on IoT device } n, \\ 0, & \text{otherwise,} \end{cases} \tag{7}$$

and

$$x_m^s(t) = \begin{cases} 1, & \text{if microservice } s \text{ is cached on EN } m, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

When the IoT device n requests microservice s, there are four ways to fetch microservice as follows.

Case 1:Local Caching

For IoT device n, if the target microservice is cached in the local cache list, i.e., $x_n^s(t) = 1$, it can be accessed directly. In this way, assume that the fetching delay $d_{n,s}^L(t)$ is zero. If the requested microservice is missed in the local storage, i.e.,$x_n^s(t) = 0$, IoT device downloads the request from the centralized cloud.

Case 2: Edge Caching

ENs with the caching capabilities also can serve several IoT devices for microservice caching. If $x_m^s(t) = 1$, IoT devices fetch the requested microservice from EN m. As a result, the fetching delay for IoT device n to download microservice s from EN m in time slot t is calculated by

$$d_{m,n,s}^E(t) = a_{m,n}^s(t) r_n^s(t) x_m^s(t) \frac{z_s}{\beta_{m,n}(t)}. \tag{9}$$

Case 3: Edge Cooperation

If the target EN misses the requested microservice s, i.e., $x_m^s(t) = 0$, the EN m_1 can find the neighboring EN m_2 for microservice s which is stored in the m_2's caching list. Specifically, m_1 finds the nearest EN m_2 that cached the requested microservice s within the communication range. And the microservice s is through m_1 to transmit from m_2 to the requested IoT device. If there is no available ENs cache s within its communication coverage, m_1 has to retrieve from the cloud. Therefore, in the cooperation mode, the fetching delay for IoT device n to download microservice s from EN m in time slot t needs additional transmission time. The transmission delay between EN m_1 and neighboring EN m_2 of microservice s is calculated by

$$d_{m1,m2}^s(t) = \frac{z_s}{\beta_{m1,m2}(t)}. \tag{10}$$

Therefore, the total fetching delay, in this case, is given by

$$d_{m,n,s}^{ECo}(t) = d_{m,n,s}^E(t) + d_{m1,m2}^s(t). \tag{11}$$

Case 4: Cloud Fetching

At the last, if the IoT device, EN, and all its neighboring ENs do not cached the requested microservice s, the IoT device will download the microservice from the remote data centers with a longer transmission latency. Therefore, the fetching delay for IoT device n to download microservice s from the cloud in time slot t is expressed by

$$d_{n,s}^{C}(t) = r_n^s(t)\frac{z_s}{\beta_{n,C}(t)}, \tag{12}$$

where $\beta_{n,C}(t)$ is the transmission rate between IoT device n and the cloud in time slot t.

Considering the delay-sensitive application requirements, we assume that each request generated from IoT devices has a deadline constraint. Let $T_n(t)$ denote the deadline constraint for IoT device n in time slot t.

Intuitively, the storage resources of IoT devices and ENs are finite to cache the part of microservices. Such that, let $\pi_n(t)$ and $\pi_m(t)$ denote the number of cached microservices of IoT device n and EN m in time slot t. Note that the cached microservices cannot exceed the caching capacity constraints, i.e.,

$$\sum_{s\in\mathcal{S}} x_n^s(t) z_s(t)\pi_n(t) \le c_n^{UE}, \forall n \in \mathcal{N}, \tag{13}$$

and

$$\sum_{s\in\mathcal{S}} x_m^s(t) z_s(t)\pi_m(t) \le c_m^{EN}, \forall m \in \mathcal{M}. \tag{14}$$

4 Problem formulation

The MEC-enabled cooperative caching problem is formulated as a MDP in this paper. For each IoT device, it observes its states from the environment. Then IoT device n performs the action to select the caching mode (i.e., at local or edge) and caching replacement decision. As a result, the reward value of each IoT is obtained based on its own state and action. Each IoT device seeks for the minimization of long-term rewards to improve the QoS. The definitions of state, action, and reward are described in detail as follows.

4.1 State

In time slot t, each IoT device n collects its observation, which includes the microservice request information, the deadline constraint, and the immediate location. Particularly, each IoT device n observes the state as follows.

$$O_n(t) = \{\mathcal{R}_n(t), z_s(t), T_n(t), x_n(t), y_n(t)\}, \tag{15}$$

where $\mathcal{R}_n(t)$ is the microservice request vector of IoT device n. Note that in each time slot, if IoT device n generate a fetching request, there is only one element in $\mathcal{R}_n(t)$ equals one (the requested microservice) and others are zero. $z_s(t)$ is the size of the requested microservice s. $T_n(t)$ is the fetching deadline of IoT device n. $x_n(t)$ and $y_n(t)$ denote the location of IoT device n in time slot t.

4.2 Action

In time slot t, each IoT device n performs the action if the request generates. The action of IoT device n includes the selection of caching mode and microservice replacement decision. Particularly, the action of each IoT device n as follows.

$$A_n(t) = \{\psi_n(t), \varphi_n(t)\}, \tag{16}$$

where $\psi_n(t)$ is the indicator whether IoT device n fetch the requested microservice from the local buffer or EN. $\psi_n(t) \in [0, M]$ is the integer variable, where $\psi_n(t) = 0$ means the local fetching, otherwise, the candidate EN is selected. $\varphi_n(t)$ is the replacement decision of IoT device n denoting which microservice be replaced with the requested one.

4.3 Reward

The objective for each IoT device n is to minimize the fetching delay. In time slot t, the delay of IoT device n is defined as follows.

$$D_n(t) = \begin{cases} d^L_{n,s}(t), \text{ if } Case1, \\ d^E_{m,n,s}(t), \text{ if } Case2, \\ d^{ECo}_{m,n,s}(t), \text{ if } Case3, \\ d^C_{n,s}(t), \text{ if } Case4. \end{cases} \tag{17}$$

Note that if the fetching delay exceeds the deadline $T_n(t)$, the reward punish is needed. Therefore, the reward is redefined as

$$R_n(t) = \begin{cases} -D_n(t), \text{ if } D_n(t) \leq T_n(t), \\ -\delta \cdot D_n(t), \text{ otherwise}, \end{cases} \tag{18}$$

where δ is the punish coefficient of float number set in manual.

5 DIMA design

In this section, the details of a DRL-based distributed cooperative microservice caching method, named DIMA, is illustrated. Specifically, the framework of the DIMA is described. Then, we present the neural network architecture of the DRL agent. At last, the training process of DRL agents on ENs and the distributed cooperative caching algorithm on IoT devices are formulated, respectively.

5.1 Framework overview

Figure 2 shows the framework of DIMA. Each IoT device independently obtains the optimal caching and microservice replacement decision through information interaction with the constructed environment. Specifically, each IoT device has a DRL agent (i.e., D3QN) and the replay buffer, which are used to store four-tuple transactions of state, action, reward, and next state. And a mini batch technology is adopted to sample some transactions from buffer to the D3QN agent for training. The IoT device obtains the state $\{\mathcal{R}_n(t), z_s(t), T_n(t), x_n(t), y_n(t)\}$ from the environment in time slot t. Through the learning of the agent, the optimal action $\{\psi_n(t), \varphi_n(t)\}$ with the largest reward value is acquired

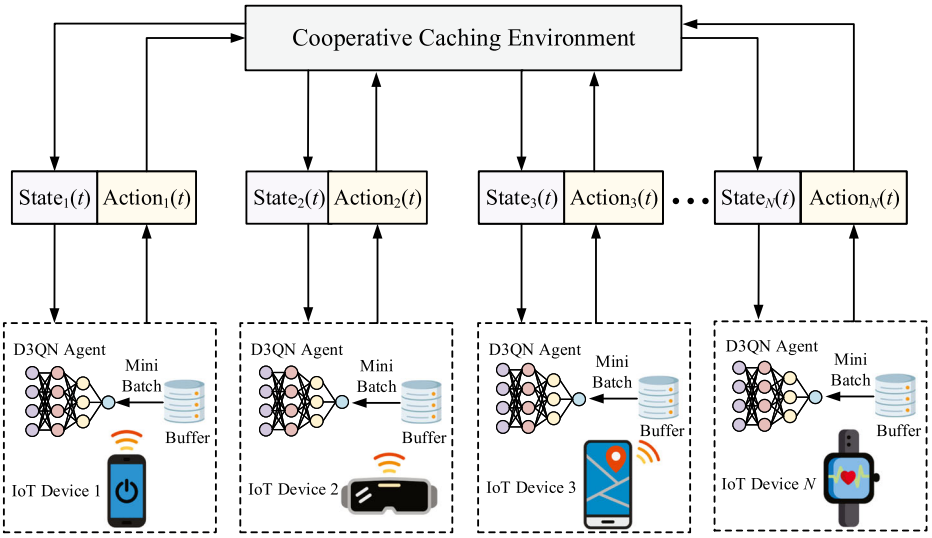

Figure 2 The framework of DIMA

to maximize long-term benefits. Particularly, considering the limited computation and storage capabilities imposed on IoT devices, the agent training and replay buffer storing of IoT devices are handled by the EN to mitigate the pressure of the resource-constrained IoT devices, which will be discussed in detail in the following.

5.2 Network architecture of DRL agent

For the formulated problem, stochastic microservice requests and the uncertainty of network communication become the challenges to solve the problem. Generally, traditional optimization methods, such as convex optimization and game theory, are difficult to obtain the optimal decisions from the stochastic environments. Reinforcement learning explores and searches the optimal strategy without any prior knowledge through the interaction with the unknown environment. Recently, DQN [17] as the first DRL algorithm that combines reinforcement learning and deep learning technology, which is suitable to solve the complex microservice caching problem in the MEC environment. However, due to the shortcomings of the high Q-value evaluation of DQN, two improved versions based on DQN algorithm have appeared, i.e., double DQN [6] and dueling DQN [32]. Therefore, in this article, we combine the double DQN and dueling DQN and propose a distributed collaborative microservice caching algorithm.

Figure 3 shows the details of the network structure of the D3QN agent. First, the input layer contains the state information of each IoT device. All states are sent as the input vectors into the D3QN network for learning, and the agent outputs the optimal action as the time slot increases. As shown in Figure 3, the input vectors are forward to the upper and lower branch networks (i.e., evaluate network and target network), respectively. Due to the overestimation of the DQN algorithm, the Q-value evaluation is generally too high, thereby reducing the performance of action selection. Inspired by the double DQN, we utilize the evaluate and target network to decouple the Q-value action selection and computation, effectively avoiding the drawbacks of the DQN algorithm. Specifically, the evaluate network is

used to select the action with the largest Q-value $A_n^{max}(t)$, while the target network calculates the target Q-value through the action selected by the evaluate network, where the target Q-value of D3QN IoT device n is calculated by

$$Y_n(t) = R_n(t+1) + \gamma \cdot Q\left(O_n(t+1),\ \text{argmax}\ Q\left(O_n(t+1); \theta_n(t)\right); \theta_n(t)\right), \tag{19}$$

where γ is the discount indicator, $\theta_n(t)$ is the parameters of network of IoT device n in time slot t. The evaluate and target both include two fully connected (FC) layers. After each FC layer, a rectified linear unit (ReLU) is used as the activation function. After the linear forward, we use the V function and the advantage function to calculate the Q-value, which is based on the dueling DQN algorithm. The core idea of dueling DQN is to use two functions (i.e., V and advantage function) based on the state and action information at the same time to conduct a more comprehensive evaluation of the Q-value and to select the more appropriate action.

Particularly, the V function evaluates the performance of the state, and the advantage function is utilized to evaluate the performance of the selected action based on state. Therefore, the modified Q-value function of $A_n(t)$ on $O_n(t)$ for IoT device n in time slot t is given by

$$\begin{aligned} Q_n\left(O_n(t), A_n(t); \theta_n(t)\right) &= V\left(O_n(t); \theta_n(t)\right) \\ &+ \left[Ad\left(O_n(t), A_n(t); \theta_n(t)\right) - \frac{1}{|\mathcal{A}|} \sum_{A_n'(t) \in \mathcal{A}} Ad\left(O_n(t), A_n'(t); \theta_n(t)\right) \right], \end{aligned} \tag{20}$$

where $V\left(O_n(t); \theta_n(t)\right)$ is the V function, $Ad\left(O_n(t), A_n(t); \theta_n(t)\right)$ is the advantage function, and $\mathcal{A}$ is the set of action space.

5.3 Training process on ENs

Considering the limited computing resources and battery life of IoT devices, in this paper we push the training of DRL agents on ENs to relieve the computing pressure of IoT devices. Meanwhile, in order to ensure the efficiency of decision-making by IoT devices, training algorithms at the edge are proposed for information exchange with IoT devices and to improve the training efficiency, where the training process algorithm at ENs is illustrated in Algorithm 1. The details of Algorithm 1 are described as follows.

In the initialization phase, EN initializes the network parameter assignment for the DRL of each IoT device (line 1). In addition, the number of episodes, discount factor γ, and batch size Ω are initialized at the beginning (lines 2). And EN initializes the replay buffer of IoT devices used to store four-tuple transactions (lines 3). Particularly, due to the limited storage capacity, we put the replay buffer of all IoT devices to ENs to alleviate the storing pressure of IoT devices. In this way, ENs can locally sample the batch of transactions of IoT device n to train its own D3QN agent. In one episode, EN transmits the network parameters of its agent for all IoT devices. The IoT device evaluates actions based on the received parameters and finally selects the action that maximizes the reward. At the same time, the IoT device stores the state, action, reward, and next state as a four-tuple, and pass it to the EN as a transaction. Therefore, if EN receives transaction $(O_n(t), A_n(t), R_n(t), O_n(t+1))$, it will be saved in the replay buffer of the corresponding IoT device n (lines 7 to 8). Specifically, the EN receives the transaction of one IoT device and performs the following procedures until the time runs out within one episode. For IoT device n, when the network training conditions are met, Ω transactions $(O_n(t), A_n(t), R_n(t), O_n(t+1))$ are sampled from the replay buffer, and the target Q-value $Y_{n,i}(t)$ of each sample i is calculated (lines 12). In order

to improve the evaluation performance of the network and update the evaluate Q network, the mean square loss function is introduced to minimize the gap between the target Q-value and the evaluated Q-value, where the loss function is defined as

$$\text{Loss}_{\,n}\left(\theta_n(t)\right) = \frac{1}{\Omega}\sum_{i=1}^{\Omega}\left[Y_{n,i}(t) - Q_n\left(O_{n,i}(t), A_{n,i}(t); \theta_n(t)\right)\right]^2. \tag{21}$$

To minimize the loss function $\text{Loss}_n\left(\theta_n(t)\right)$, the backpropagation operation is utilized with the gradient descent algorithm (lines 14). Moreover, to achieve the optimal performance, it is essential to analyze the convergence of the (21). To our knowledge, there are many works that have focused on the convergence analysis of the (21) and presented the mathematical proof, e.g., [31]. In addition, when a certain number of training times is reached, the target network parameter $\theta_n(t)'$ is replaced by the updated network parameter $\theta_n(t)$. Therefore, through the iterative process, network parameters are continuously optimized, such that the DRL agent can provide IoT devices with optimal action decision-making.

Algorithm 1 Training algorithm for D3QN agents.

```
1  Initialize the evaluate Q and target Q network with the parameter θ_n(t) and θ_n(t)', and
   set θ_n(t) → θ_n(t)';
2  Initialize the number of episodes, discount indicator γ, and batch size Ω;
3  Initialize the replay buffer for each IoT device n;
4  for each episode do
5      for each IoT device n ∈ 𝒩 do
6          Transmit the network parameters to IoT device n;
7          while receive the transaction do
8              Store the transaction into replay buffer of n;
9              if execute the network training then
10                 Sample Ω transactions from IoT device n;
11                 for each transaction do
12                     Calculate the target Q-value Y_n(t) by (19)
13                 end
14                 Minimize the loss value by (21) based on Ω samples and update the
                   parameters
15             end
16         end
17     end
18 end
```

5.4 Distributed cooperative caching algorithm on IoT devices

To alleviate computing and storing pressures in the IoT devices, the agent training and replay buffer storing are handled by the ENs. In this way, each IoT device enables to execute actions independently without any prior knowledge through the interaction of ENs. To this

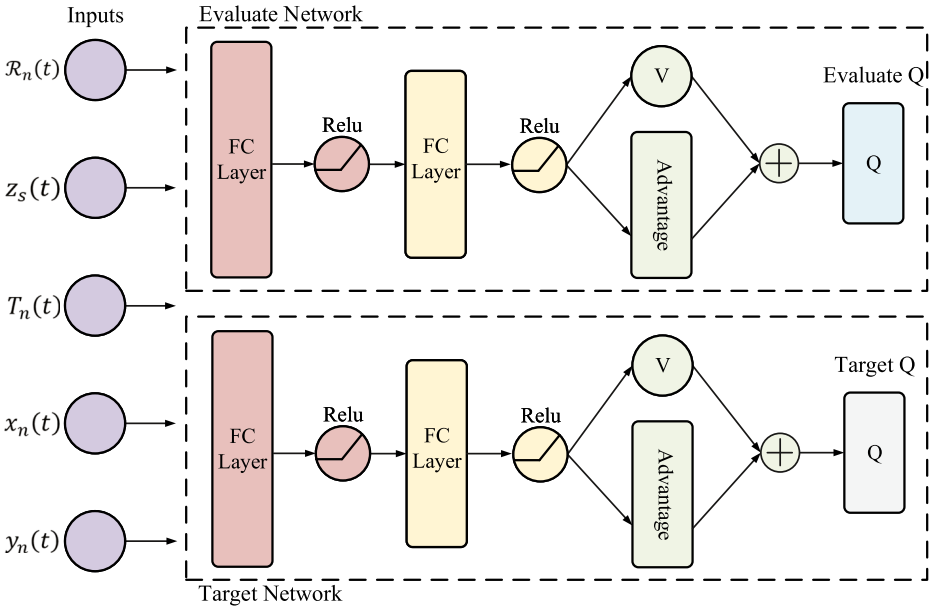

Figure 3 An illustration of network architecture of D3QN

end, we propose a distributed cooperative microservice caching algorithm based on D3QN for each IoT device n to perform the caching mode selection and microservice replacement in a decentralized manner, which is presented in Algorithm 2. The detailed descriptions are as follows.

As shown in Algorithm 2, each IoT device performs the DRL-based collaborative microservice caching decisions. Specifically, IoT device n sets the number of episodes in the initialization phase at the beginning (lines 1). In one episode, the constructed environment is first to reset, and a loop of T time slots is executed. In each time slot t, IoT device n obtains state information $\{\mathcal{R}_n(t), z_s(t), T_n(t), x_n(t), y_n(t)\}$ at first (lines 5). According to the obtained state and the network parameters received from the EN (lines 6), the ϵ-greeedy method is used to select actions, where the $A_n(t)$ is given by

$$A_n(t) = \begin{cases} \text{random choice from the action space, if } rand < \epsilon, \\ \text{argmax } Q\left(O_n(t+1); \theta_n(t)\right), \text{ otherwise,} \end{cases} \tag{22}$$

where $rand$ is a float number from 0 to 1, ϵ is a probability, and $argmax\ Q(O_n(t+1); \theta_n(t))$ means an action of the maximum Q-value. According to the selected action, the $R_n(t)$ and next state $O_n(t+1)$ are derived (lines 8). As a result, IoT device n have to send the collected four-tuple transaction $(O_n(t), A_n(t), R_n(t), O_n(t+1))$ to EN for the operation of Algorithm 1.

Algorithm 2 Distributed cooperative microservice caching algorithm for IoT device $n \in \mathcal{N}$.

```
Initialize the number of episodes and time slot;
for each episode do
    Reset the environment;
    for each time slot t ∈ 𝒯 do
        Observe the O_n(t);
        Receive the network parameters θ_n(t) from EN;
        Choose the A_n(t) by (22);
        Obtain the R_n(t) and next state O_n(t + 1);
        Send the four-tuple transaction (O_n(t), A_n(t), R_n(t), O_n(t + 1)) to EN;
    end
end
```

6 Simulation results

In this section, extensive simulation experiments are conducted to evaluate the performance of the DIMA. Particularly, the experimental setup is described first. Then, the performance evaluation is presented. All the simulations are conducted on a PC Server of Manjaro Linux 21.1.0 with Intel i5-10600KF (4.10GHz), 64GB RAM, and 2 NVIDIA RTX3090 GPUs. Python 3.9.5 with Pytorch 1.8.1 is employed as the experimental environment.

6.1 Experimental setups

At first, the simulation is conducted in a 500m $\times$ 500m range area with 10 ENs and 10 IoT devices. For each IoT device, it has one DRL agent to interact with the environment and perform action independently without the complete information of other IoT devices. The microservice requests are generated based on the Zipf distribution with 10000 microservices. The transmission radius of EN is set to 75 meters. The transmission power is modeled as 0.25 W and Gauss white noise power is 1e-10 W. The main parameters in the simulation are illustrated in Table 1.

6.2 Baseline approaches

To evaluate the performance of the DIMA, three baselines are selected for the comparison as follows.

- Non-cooperative algorithm (Non-Cooperative). In this scheme, each EN only fetches the cached microservice without the edge cooperation. If the request misses, EN requires the cloud to download the target microservice.
- Least recently used algorithm (LRU) [21]. In this scheme, each IoT device randomly selects the caching mode and the least recently used microservice is replaced by the requested one.
- First input first output algorithm (FIFO) [27]. In this scheme, each IoT device randomly selects the caching mode and the earliest stored microservice is replaced by the requested one.

Table 1 Main Parameters

Parameter	Value
Coverage	$500 \times 500\ m^2$
Number of IoT devices	10
Number of ENs	10
Number of microservices	10000
Storage capacity of ENs	2000
Storage capacity of IoT devices	300
Bandwidth of ENs	20 MHz
Size of microservices	[5, 10] MB
Punish coefficient	1.1
Number of episodes	1e3
Learning rate	1e-3
Size of replay buffer	3000
Discount indicator	0.99
Probability of ϵ	0.9
Batch size	64

In addition, two key metrics are used to evaluate the performance of the DIMA, i.e., average fetching delay and cache hit ratio.

6.3 Performance evaluation

6.3.1 Effectiveness comparison on number of microservices

Figure 4 presents the average latency performance for different numbers of microservices. It can be seen that the average latency increases as the total number of microservices increases. This is because when there are more and more microservices, the limited local and edge cache capacity cannot cache enough microservices, which makes it more difficult to cache hits. For example, when the total number of microservices is 10000, the DIMA is 19%, 4.2%, and 3.7% lower in average latency than Non-Cooperative, LRU, and FIFO, respectively. Through the joint decision of caching mode and replacement strategy, we can see that the DIMA has better performance in different amounts of microservices. Specifically, when the number of microservices increased from 4000 to 12000, the performance improvement of DIMA compared to Non-Cooperative, LRU, and FIFO ranged from 37%, 3.8%, and 3.7% to 16%, 6.1%, and 4.6%.

Figure 5 shows the hit ratio performance evaluation under different numbers of microservices. It can be seen that with the increase of the total number of microservices, the difficulty of cache hits is also increasing. In particular, when the total number of microservices is 12000, the hit ratio of the DIMA, Non-Cooperative, LRU, and FIFO are 42.2%, 39.3%, 38.3%, and 31.7%, respectively. The DIMA has better hit ratio performance. This is because the DRL agent of each IoT device interacts with the environment to make the best choice of the caching and replacement strategies without knowing the global information. As the total number of microservices increases, the difficulty of caching increases. LRU and FIFO algorithms only replace the microservices based on historical request records, while DRL agents choose the actions with greater rewards after continuous exploration.

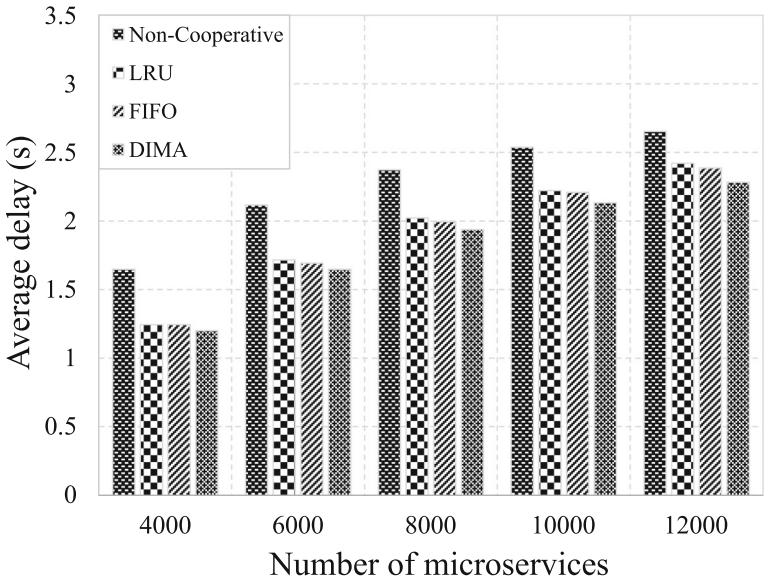

Figure 4 Average delay on different number of microservices

6.3.2 Effectiveness comparison on edge caching capacity

Figure 6 illustrates the average latency performance under different EN caching capacities. It can be seen that when the EN cache capacity becomes larger, the average fetching delay is reduced. The main reason is that more and more microservices can be stored by

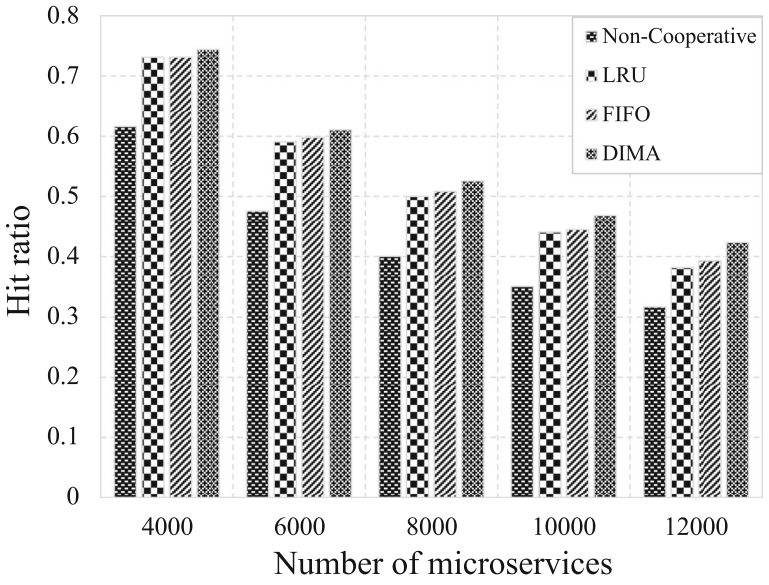

Figure 5 Hit ratio on different number of microservices

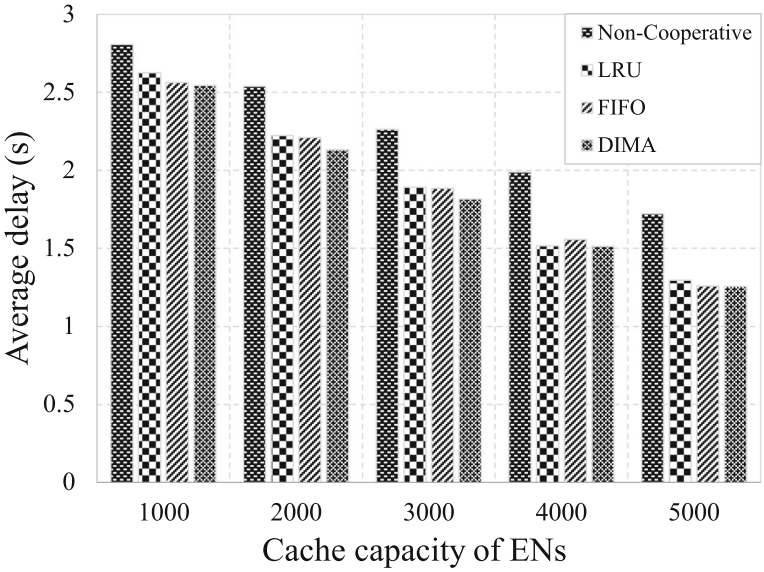

Figure 6 Average delay on different cache capacity of ENs

EN. Under the constant number of microservices, the storage capacity of EN has become the key to determining the caching performance. In particular, when the EN cache capacity is 1000, the DIMA has a 10% lower latency than the Non-Cooperative algorithm. However, when the cache capacity is 5000, the DIMA is 37% lower than the Non-Cooperative. Larger cache capacity means that more microservices are cached when performing edge cooperation caching, which greatly reduces the retrieving time.

Figure 7 describes the hit ratio performance evaluation of different EN cache capacities. It can be seen that when the cache capacity increases from 1000 to 5000, the DIMA and LRU and FIFO increase from about 30% to more than 70%. However, the best hit ratio for Non-Cooperation is only 59% when the cache capacity is 5000. This is because the increase in caching capacity enables all ENs to cache more microservices. Therefore, even when an EN fails to hit the request target, its neighboring nodes are more likely to cache the request, which reduces the high transmission delay of obtaining microservices from the cloud after the miss. In addition, the DIMA has an average 4% and 2.5% improvement over LRU and FIFO in terms of hit ratio performance when the cache capacity is from 1000 to 5000. Therefore, the DIMA can select a better caching mode and microservice replacement strategy.

6.3.3 Effectiveness comparison on bandwidth

Figure 8 shows the average fetching delay performance evaluation under different bandwidth resources. It can be seen that the more bandwidth resources of the EN, the better delay performance can be obtained. For example, when the bandwidth is 10MHz, the DIMA is 17.7%, 3.2%, and 1.4% lower than Non-Cooperative, LRU, and FIFO in average delay, respectively. The main reason is that more bandwidth ensures faster transmission between IoT devices and EN and between EN and EN, which can effectively reduce the delay in obtaining microservices. When the bandwidth is 30MHz, the average delay of the DIMA

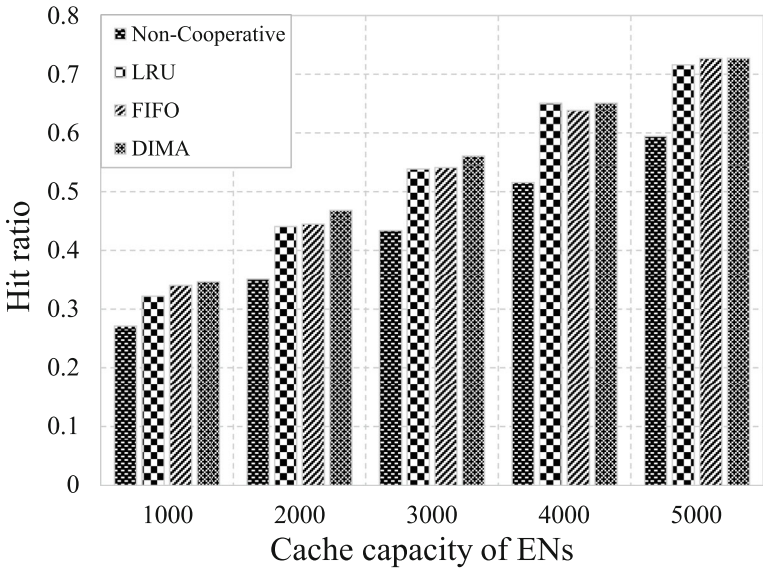

Figure 7 Hit ratio on different cache capacity of ENs

is 17.2, 1.5%, and 1.1% lower than Non-Cooperative, LRU, and FIFO. It can be seen that the increase of EN bandwidth resources is not obvious to the improvement of the delay performance difference between the DIMA and the baseline algorithms.

Figure 9 shows the hit ratio performance under different bandwidth resources. It can be seen that with the continuous increase of bandwidth on ENs, the hit ratio of the DIMA, non-cooperative, LRU and FIFO has not changed significantly. On average, the DIMA has

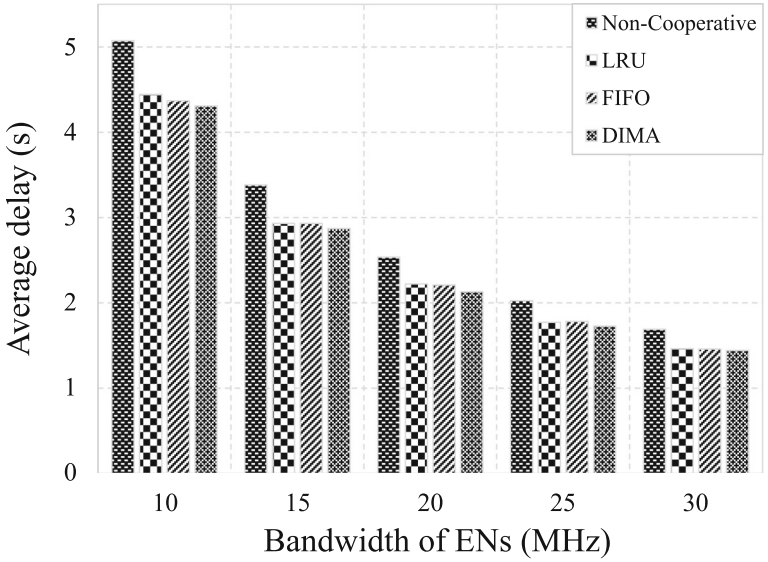

Figure 8 Average delay on different bandwidth of ENs

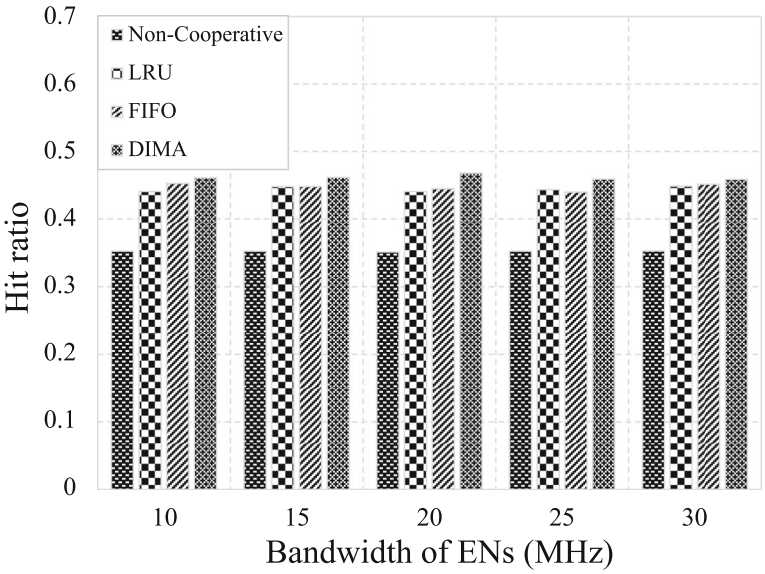

Figure 9 Hit ratio on different bandwidth of ENs

31%, 4% and 3.2% improvement than Non-Cooperative, LRU and FIFO on hit ratio. This is because changes in bandwidth resources mainly affect the transmission latency between IoT devices and EN, and between EN and EN, which cannot directly influence the performance of the hit ratio. However, when considering the contention of bandwidth resources on the EN, it is necessary to evaluate the bandwidth resources of the selected EN to make an optimal decision of EN selection.

6.3.4 Effectiveness comparison on Zipf parameter

Figure 10 illustrates the average delay performance evaluation under different Zipf distribution parameter α. In the experiment, we assumed that the access requests of microservices follow the Zipf distribution, which means that the frequency of appearance of each microservice is inversely proportional to its request ranking. For the requests that follow the Zipf distribution, most of the access are focused on a few microservices. It can be seen that as the parameter α continues to increase, the delay decreases. This is because the parameter α determines the density of the Zipf distribution. The larger the α, the denser the distribution. Otherwise, the distribution is sparse. For example, when the parameter α is 0.3, the DIMA reduces the latency by 15.8%, 0.5%, and 0.8% compared with Non-Cooperative, LRU, and FIFO. As the parameter increases to 0.9, the decrease of the DIMA compared with Non-Cooperative, LRU and FIFO is 19%, 4.2%, and 3.7%, respectively.

Figure 11 expresses the hit ratio performance under different Zipf distribution parameter α. We can see that as α increases, the hit ratio increases. For example, when α is 0.3, the hit ratio of the DIMA, Non-Cooperative, LRU, and FIFO are 32.3%, 20.5%, 31.9%, and 31.8%, respectively. As the Zipf distribution parameter α increases to 0.9, the hit ratio

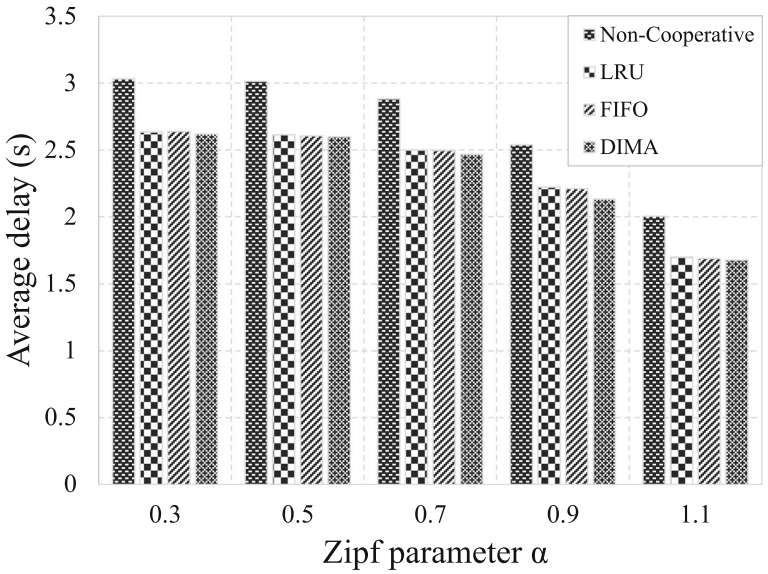

Figure 10 Average delay on different Zipf Parameter α

of the DIMA, Non-Cooperative, LRU, and FIFO increase to 46.8%, 35.1%, 44.1%, and 44.4%, respectively. This is because, as α increases, the distribution of access requests is increasingly gathered in a few microservices. Therefore, when some microservices appear with a higher request frequency, the cache hit ratio increases accordingly.

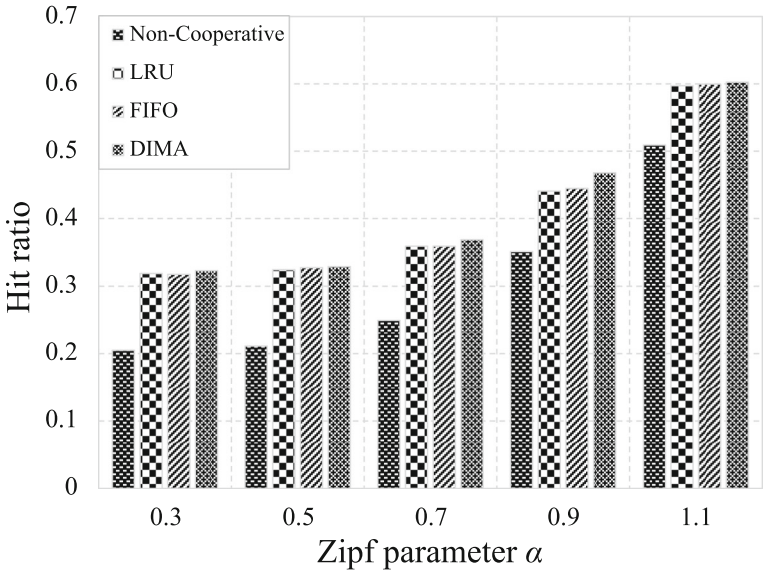

Figure 11 Hit ratio on different Zipf Parameter α

7 Conclusion

In this paper, we investigated the problem of cooperative microservice caching in MEC-enabled IoT systems and proposed a distributed cooperative microservice caching scheme, DIMA. To minimize the microservice retrieval delay and improve the cache hit ratio, the cooperative microservice caching problem was modeled as a MDP. For the formulated MDP, we proposed a distributed DRL-based caching algorithm with the cooperation between ENs, which enables each IoT device performs action independently with no complete global information. Moreover, to mitigate the computation pressure of IoT devices, we proposed a training algorithm to enable the training of DRL agents at ENs which interact with IoT devices to transmit the intermediate information (i.e., network parameters and four-tuple transactions). Eventually, extensive simulations based on Python with Pytorch are conducted and numerical results are demonstrated that the DIMA is more well-performed than baselines in average fetching delay and hit ratio.

Acknowledgements This research is supported by the Financial and Science Technology Plan Project of Xinjiang Production and Construction Corps under grant no. 2020DB005. This research is also supported by the National Natural Science Foundation of China under grant no.41975183 and 62173023.

References

1. Bi, R., Liu, Q., Ren, J., Tan, G.: Utility aware offloading for mobile-edge computing. Tsinghua Sci. Technol. **26**(2), 239–250 (2020)
2. Bi, S., Huang, L., Zhang, Y.-J.A.: Joint optimization of service caching placement and computation offloading in mobile edge computing systems. IEEE Trans. Wirel. Commun. **19**(7), 4947–4963 (2020)
3. Chen, H., Zhang, Y., Cao, Y., Xie, J.: Security issues and defensive approaches in deep learning frameworks. Tsinghua Sci. Technol. **26**(6), 894–905 (2021)
4. Chen, L., Song, L., Chakareski, J., Jie, X.: Collaborative content placement among wireless edge caching stations with time-to-live cache. IEEE Trans. Multimed. **22**(2), 432–444 (2019)
5. Chen, S., Yao, Z., Jiang, X., Yang, J., Hanzo, L.: Multi-agent deep reinforcement learning-based cooperative edge caching for ultra-dense next-generation networks. IEEE Trans. Commun **69**(4), 2441–2456 (2020)
6. Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence AAAI'16, pp. 2094–2100. AAAI Press (2016)
7. He, Q., Cui, G., Zhang, X., Chen, F., Deng, S., Jin, H., Li, Y., Yang, Y.: A game-theoretical approach for user allocation in edge computing environment. IEEE Trans. Parall. Distrib. Syst. **31**(3), 515–529 (2019)
8. Hong, Z., Chen, W., Huang, H., Guo, S., Zheng, Z.: Multi-hop cooperative computation offloading for industrial iot–edge–cloud computing environments. IEEE Trans. Parall. Distrib. Syst. **30**(12), 2759–2774 (2019)
9. Hu, S., Guanghui, L.: Dynamic request scheduling optimization in mobile edge computing for iot applications. IEEE Internet Things J. **7**(2), 1426–1437 (2019)
10. Jaramillo, D., Nguyen, D.V., Smart, R.: Leveraging microservices architecture by using docker technology. In: SoutheastCon 2016, pp. 1–5. IEEE (2016)
11. Kim, J., Kim, T., Hashemi, M., Brinton, C.G., Love, D.J.: Joint optimization of signal design and resource allocation in wireless d2d edge computing. In: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, pp. 2086–2095. IEEE (2020)
12. Lei, L., Xu, H., Xiong, X., Zheng, K., Xiang, W.: Joint computation offloading and multiuser scheduling using approximate dynamic programming in nb-iot edge computing systemxc. IEEE Internet Things J. **6**(3), 5345–5362 (2019)
13. Li, S., Li Da, X., Zhao, S.: 5g internet of things A survey. J. Industr. Inform. Integr. **10**, 1–9 (2018)
14. Liu, Y., Zeng, Z., Liu, X., Zhu, X., Bhuiyan, M.Z.A.: A novel load balancing and low response delay framework for edge-cloud network based on sdn. IEEE Internet Things J. **7**(7), 5922–5933 (2019)

15. Mabrouki, J., Azrour, M., Dhiba, D., Farhaoui, Y., Hajjaji, S.E.: Iot-based data logger for weather monitoring using arduino-based wireless sensor networks with remote graphical application and alerts. Big Data Mining Analytics **4**(1), 25–32 (2021)
16. Malek, Y.N., Najib, M., Bakhouya, M., Essaaidi, M.: Multivariate deep learning approach for electric vehicle speed forecasting. Big Data Mining Analytics **4**(1), 56–64 (2021)
17. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv:1312.5602 (2013)
18. Ning, Z., Dong, P., Wang, X., Xiping, H., Guo, L., Bin, H., Yi, G., Qiu, T., Kwok, R.YK.: Mobile edge computing enabled 5g health monitoring for internet of medical things A decentralized game theoretic approach. IEEE J. Select. Areas Commun. **39**(2), 463–478 (2021)
19. Prabadevi, B., Deepa, N., Pham, Q.-V., Nguyen, DC., Praveen Kumar Reddy, M., Thippa Reddy, G., Pathirana, P.N., Dobre, O.: Toward blockchain for edge-of-things A new paradigm, opportunities, and future directions. IEEE Internet Things Magaz. **4**(2), 102–108 (2021)
20. Pingyang, W., Li, J., Shi, L., Ding, M., Cai, K., Yang, F.: Dynamic content update for wireless edge caching via deep reinforcement learning. IEEE Commun. Lett. **23**(10), 1773–1777 (2019)
21. Qi, H., Birman, K., Renesse, R.V., Lloyd, W., Kumar, S., Li, H.C.: An analysis of facebook photo caching. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, SOSP '13, pp 167–181 (2013)
22. Rafique, W., Qi, L., Yaqoob, I., Imran, M., Rasool, R.U., Dou, W.: Complementing iot services through software defined networking and edge computing: A comprehensive survey. IEEE Commun. Surv. Tutor. **22**(3), 1761–1804 (2020)
23. Ren, Z., Liu, Y., Shi, T., Xie, L., Zhou, Y., Zhai, J., Zhang, Y., Zhang, Y., Chen, W.: Aiperf: Automated machine learning as an ai-hpc benchmark. Big Data Mining Analytics **4**(3), 208–220 (2021)
24. Samanta, A., Tang, J.: Dyme: Dynamic microservice scheduling in edge computing enabled iot. IEEE Internet Things J. **7**(7), 6164–6174 (2020)
25. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: Vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)
26. Song, C., Xu, W., Wu, T., Yu, S., Zeng, P., Zhang, N.: Qoe-driven edge caching in vehicle networks based on deep reinforcement learning. IEEE Trans. Veh. Technol. **70**(6), 5286–5295 (2021)
27. Tang, L., Huang, Q., Lloyd, W., Kumar, S., Li, K.: {RIPQ}: Advanced photo caching on flash for facebook. In: 13th {USENIX} Conference on File and Storage Technologies ({FAST} 15), pp. 373–386. USENIX Association (2015)
28. Tong, Z., Ye, F., Yan, M., Liu, H., Basodi, S.: A survey on algorithms for intelligent computing and smart city applications. Big Data Mining Analytics **4**(3), 155–172 (2021)
29. Wang, Y., He, Q., Ye, D., Yang, Y.: Formulating criticality-based cost-effective fault tolerance strategies for multi-tenant service-based systems. IEEE Trans. Softw. Eng. **44**(3), 291–307 (2017)
30. Wang, S., Guo, Y., Zhang, N., Yang, P., Zhou, A., Shen, X.S.: Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach. IEEE Trans. Mobile Comput. **20**(3), 939–951 (2021)
31. Wang, X., Li, R., Wang, C., Li, X., Taleb, T., Leung, V.ictor.C.M.: Attention-weighted federated deep reinforcement learning for device-to-device assisted heterogeneous collaborative edge caching. IEEE J. Select. Areas Commun. **39**(1), 154–169 (2020)
32. Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., Freitas, N.: Dueling network architectures for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1995–2003. PMLR (2016)
33. Wei, D., Ning, H., Shi, F., Wan, Y., Xu, J., Yang, S., Zhu, L.: Dataflow management in the internet of things: Sensing, control, and security. Tsinghua Sci. Technol. **26**(6), 918–930 (2021)
34. Xia, X., Chen, F., He, Q., Grundy, J.C., Abdelrazek, M., Jin, H.: Cost-effective app data distribution in edge computing. IEEE Trans. Parall. Distrib. Syst **32**(1), 31–44 (2020)
35. Xie, Z., Chen, W.: Storage-efficient edge caching with asynchronous user requests. IEEE Trans. Cognit. Commun. Netw. **6**(1), 229–241 (2019)
36. Xiong, X., Zheng, K., Lei, L., Hou, L.u.: Resource allocation based on deep reinforcement learning in iot edge computing. IEEE J. Select. Areas Commun. **38**(6), 1133–1146 (2020)
37. Xu, X., Xihua, L., Xu, Z., Fei, D., Xuyun, Z., Lianyong, Q.: Trust-oriented iot service placement for smart cities in edge computing. IEEE Internet Things J. **7**(5), 4084–4091 (2019)
38. Xu, Z., Wang, S., Liu, S., Dai, H., Xia, Q., Liang, W., Wu, G.: Learning for exception: Dynamic service caching in 5g-enabled mecs with bursty user demands. In: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), pp. 1079–1089. IEEE (2020)
39. Zhang, K., Cao, J., Liu, H., Maharjan, S., Zhang, Y.: Deep reinforcement learning for social-aware edge computing and caching in urban informatics. IEEE Trans. Industr. Informat. **16**(8), 5467–5477 (2019)

40. Zhang, T., Fang, X., Liu, Y., Li, G.Y., Xu, W.: D2d-enabled mobile user edge caching: A multi-winner auction approach. IEEE Trans. Veh. Technol. **68**(12), 12314–12328 (2019)
41. Zhang, W., Chen, X., Jiang, J.: A multi-objective optimization method of initial virtual machine fault-tolerant placement for star topological data centers of cloud systems. Tsinghua Sci. Technol. **26**(1), 95–111 (2020)
42. Zhang, Y., Meng, L., Xue, X., Zhou, Z., Tomiyama, H.: Qoe-constrained concurrent request optimization through collaboration of edge servers. IEEE Internet Things J. **6**(6), 9951–9962 (2019)
43. Zhong, C., Gursoy, M., Velipasalar, S.: Deep reinforcement learning-based edge caching in wireless networks. IEEE Trans. Cogn. Commun. Netw. **6**(1), 48–61 (2020)
44. Zhou, X., Liang, W., She, J., Yan, Z., Wang, K.I.-K.: Two-layer federated learning with heterogeneous model aggregation for 6g supported internet of vehicles. IEEE Trans. Veh. Technol. **70**(6), 5308–5317 (2021)
45. Zhou, X., Liang, W., Shimizu, S., Ma, J., Jin, Q.: Siamese neural network based few-shot learning for anomaly detection in industrial cyber-physical systems. IEEE Trans. Industr. Inform. **17**(8), 5790–5798 (2020)

Affiliations

Hao Tian[1] · Xiaolong Xu[1,2,3,4,5] · Tingyu Lin[6] · Yong Cheng[7] · Cheng Qian[8] · Lei Ren[9] · Muhammad Bilal[10]

Hao Tian
haotian@nuist.edu.cn

Tingyu Lin
lintingyu2003@sina.com

Yong Cheng
yongcheng@nuist.edu.cn

Cheng Qian
chengqian-research@outlook.com

Lei Ren
renlei@buaa.edu.cn

Muhammad Bilal
m.bilal@ieee.org

1 School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing 210044, China

2 Engineering Research Center of Digital Forensics, Ministry of Education, Nanjing University of Information Science and Technology, Nanjing, China

3 Jiangsu Collaborative Innovation Center of Atmospheric Environment and Equipment Technology (CICAEET), Nanjing University of Information Science and Technology and Engineering, Nanjing 210044, China

4 Provincial Key Laboratory for Computer Information Processing Technology, Soochow University, Suzhou, China

5 State Key Laboratory Novel Software Technology, Nanjing University, Nanjing, China

6 State Key Laboratory of Complex Product Intelligent Manufacturing System Technology, Beijing Institute of Electronic System Engineering, Beijing, China

7 School of Automation, Nanjing University of Information Science and Technology, Nanjing 210044, China

8 Jiangsu Hydraulic Research Institute, Nanjing 210017, China

9 School of Automation Science and Electrical Engineering, Beihang University, Beijing, China

10 Depatment of Computer and Electronics Systems Engineering, Hankuk University of Foreign Studies, Yongin-si, Gyeonggi-do, 17035, Korea