Computing Offloading with Fairness Guarantee: A Deep Reinforcement Learning Method

Hao Hao, Changqiao Xu, *Senior Member IEEE,* Wei Zhang, Shujie Yang, and Gabriel-Miro Muntean, *Fellow IEEE*

***Abstract*—Edge computing can reduce service latency and save backhaul bandwidth by completing services at network edges, providing support for diverse computation-intensive and delay-sensitive services. However, it is not practical to support all services at edge nodes due to the limited network resources. The decision that which services can be provided locally and which services should been offloaded to cloud significantly impacts the user experience. Cloud-edge computing offloading becomes an important issue in edge computing. In this paper, we take the fairness into the optimization objective of computing offloading problem, and consider both computing capacity and storage space as problem constraints. The problem is formulated as a long-term average optimization problem to maximize the α-fair utility function of saved time, and further translated as a Markov decision process. As the optimization problem with fairness guarantee and huge action space, we cannot solve it with traditional methods. Therefore, an innovative multi-update deep reinforcement learning algorithm is proposed which can optimize the objective with α-fair utility function and reduce dramatically the size of action space. We also prove the convergence of our algorithm theoretically. To our best knowledge, the long-term average optimization of computing offloading with fairness guarantee is rarely seen in literature. Extensive simulation experiments show that our algorithm can converge quickly and has better performance in terms of service delay and fairness.**

***Index Terms*—Mobile Edge Computing, Computing Offloading, Deep Reinforcement Learning, Fairness Guarantee.**

I. Introduction

OUR life has been greatly enriched by the rapid development of both high specification devices and mobile networks. Many new computational-demanding services, including involving natural language processing, live rich media streaming, etc. have sprung up. These services enrich our life, but also bring exponentially growing data to process, which requires huge computing resources. It is becoming inadequate to respond such a large computational pressure with cloud computing solutions that only rely on the computing resources of cloud services. Besides, users can only get services from remote cloud in cloud computing, resulting in unacceptable network latency and congested backbone networks. To alleviate the computation pressure in the cloud, edge computing [1], a new computing paradigm, was proposed to utilize edge node resources to support some services. This approach reduces network latency by avoiding the long transmission from users to cloud services, and is gradually becoming an essential avenue to improve the quality of network services.

However, the capability of edge node is limited and they cannot satisfy all service requests currently. To take full advantage of cloud computing and edge computing, it is necessary to cooperate the resources of cloud and edge node. One of important problems is how to determine the computing status. There are many works that study this problem and have achieved excellent results in some respects [2]-[10]. However, they all ignore the fairness of services, which results in consistently short latency for some popular services but poor experience for other services. Besides, it is common to cache relevant databases/libraries before processing computational-demanding services. For example, in Dynamic Adaptive Streaming over HTTP (DASH) [11], we need to analyze packet loss, available bandwidth and other factors of users to provide appropriate bitrate versions. Before the computing, the relevant databases/libraries of applications need to be cached in edge node. Another example involves cloud games. Users offload computing tasks to edge nodes or the cloud to reduce the computing pressure. Apart from data processing, the edge node or the cloud also need to cache relevant data of the games. These show the storage space is also an important constraint for the cooperation between cloud and edge node.

In our work, we focus on the long-term average performance optimization of service offloading with fairness guarantee under the constraints of limited computing capacity and storage space at the edge node. There are many new challenges to solve this problem. First, we all know that network aspects such as user requests are dynamic and stochastic [12]. Compared with short-term performance, the long-term average performance (e.g. service latency) is also more useful. However, we usually need the complete future information to achieve the long-term average optimization, but predicting future information in a dynamic network is difficult, which means that optimization of long-term average performance is also very challenging. In addition, we need to take the fairness into account, making the problem intractable. In fact, some works have optimized the current computing offloading decision by taking fairness into account, but few works achieve the long-term average optimization with fairness guarantee.

H. Hao, W. Zhang are with the Shandong Computer Science Center (National Supercomputing Center in Jinan), Qilu University of Technology (Shandong Academy of Sciences), Jinan, China. (e-mail: haoh@sdas.org, wzhang@sdas.org)

C. Xu, S. Yang are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. (e-mail: cqxu@bupt.edu.cn, sjyang@bupt.edu.cn).

G.-M. Muntean is with the Performance Engineering Laboratory, School of Electronic Engineering, Dublin City University, Dublin, Ireland. (e-mail: gabriel.muntean@dcu.ie).

Corresponding author: Changqiao Xu

Second, the problem complexity will increase with double constraints. In the service offloading problem, both the computing capacity and storage space of the edge node affect the offloading policy, so we should address them jointly. There is a need for adaptive and efficient coordination to reduce service latency. Third, we need to get the computing status of services rather than the computing offloading ratio. However, the computing statuses of services are interrelated, resulting in an exponentially growing solution space, which means it is not feasible to search the entire solution space.

Recently, deep reinforcement learning (DRL) [13], good at long-term optimization problem solving, was widely used in control. Through training on historical data, DRL can take appropriate actions to get the optimal long-term average reward, which can help solving our problem. However, we can not use DRL directly due to the optimization objective of our problem with fairness guarantee. Additionally, there are some other issues, especially as model training takes a long time or converges slowly if the action space is very large.

This paper focuses on achieving long-term average optimization of cloud-edge computing offloading with fairness guarantee. The article formulates the problem as a Markov Decision Process (MDP) and proposes a novel DRL algorithm to solve it. The proposed solution is a significant extension of our previous work [14] as it guarantees fairness. To the best our knowledge, this is one of the very few works which focus on the long-term average optimization of computing offloading with fairness guarantee. The main contributions of our research are summarized as follows:

- To minimize service delay, **the computing offloading problem with optimization of the long-term average performance is formulated**. We also introduce the -fair utility function into our optimization objective to guarantee service fairness. Furthermore, the optimization is formulated into a Markov Decision Process (MDP) problem.
- To optimize the -fair utility function of average service delay, **we adjust the MDP with a function of past rewards.** We also theoretically prove that the adjusted model has the same stationary point as the original -fair utility function problem model.
- Due to the unavailability of state transition probability, we propose **a novel model-free deep reinforcement learning algorithm, the multi-update reinforcement learning (MURL)** to solve this problem. MURL has an adjusted reward and employs an innovative exploration strategy, which can optimize the -fair utility function of average service delay and reduce the action space size.
- **Extensive simulation experiments compare the proposed solution with three other alternative solutions.** Better results are achieved in terms of both service latency and fairness during performance evaluations.

The paper is organized as follows. The related works are reviewed in Section II. The system model is introduced in Section III. The service offloading problem is formulated as a MDP in Section IV. The novel DRL algorithm is introduced in Section V. Section VI discusses the simulation results and Section VII presents the conclusions.

II. RELATED WORKS

Mobile edge computing is attracting widespread attention and being applied to various scenarios, including transcoding for livecast services [15], software defined networking (SDN) [16], information centric networking (ICN) [17] and 5G environments [18]. Due to the limited resources of edge nodes, the collaboration between cloud and edge is necessary, which has attracted many researches and development efforts.

One of the essential aspects to address is computing offloading which will significantly affect the performance of services. Various works have focused on how to improve service quality under the limited computing capacity of edge nodes. To meet the demand of user services, authors [2] proposed a resource allocation framework for mobile-edge cloud, which combines the limited communication and computing resources. The resource optimization of computing is formulated as a mix integer nonlinear programming problem in [3] to minimize the computing time of all services. In order to adapt to the dynamic environments, the authors of [4] proposed a service offloading method based on meta reinforcement learning. This method improves the speed of training by employing a clipped surrogate objective and also reduces the delay. Authors of [5] considered the trust risks in computing offloading. They formulated a long-term optimization problem with co-provisioning of computation, transmission and trust services. Based on a Lyapunov optimization, they proposed an online learning-aided cooperative offloading mechanism to solve the problem. Considering the spectrum access in reconfigurable wireless networks, authors of [6] proposed a primary-prioritized recurrent deep reinforcement learning algorithm for dynamic spectrum access based on the cognitive radio (CR) technology. They formulated user states as Markov states. To improve the convergence speed, authors combined dueling deep Q-network with recurrent neural network to solve the problem. Authors of [7] took both energy consumption and service delay into account. They formulated a service offloading problem with the goal of minimizing the weighted sum of energy consumption and computation latency. Authors of [8] proposed a D2D-assisted MEC system whose goal is to reduce energy consumption and delay. They separately solved the computing offloading and transmission power allocation by a Knapsack problem-based algorithm and convex optimization. Finally, they proposed an alternate optimization algorithm to achieve the joint optimization of these two aspects. In Internet of Things (IoT), authors of [9] proposed an application-deadline-aware computing offloading strategy which employs deep reinforcement learning to reduce the energy consumption of IoT devices. In internet of vehicles, authors [10] formulated the service placement problem as a binary integer linear programming problem whose goal is the optimization of service delay. Then authors developed a low complexity heuristic solution to this problem. However, all these works ignore the fairness of services which is important network objective. Besides, there is an implicit assumption that edge node can process all computation tasks that are offloaded from users

regardless of whether it has already cached related data. This is impractical as there is limited edge storage space available.

Edge caching localizes traffic and achieves low latency. The authors of [19] proposed an edge caching framework for 5G networks. Based on user demands, this framework caches most popular content to minimize the average access delay. By learning users' preferences for video topics, authors [20] proposed a novel caching policy. Once receiving a service request, an explore-and-exploit method is applied to decide whether to cache the service based on users' preference. Authors of [21] studied the binary offloading avenues for AR applications. They formulated the problem as a Markov decision process and proposed a deep reinforcement learning model to solve it, which greatly reduced the computational complexity of the solution. Authors [22] designed a caching system which supports both edge computing and hierarchical caching. They formulated a hierarchical collaborative caching problem with the goal of minimizing transmission latency, and then proposed an online algorithm to solve it. Collaborative content caching between cloud and single base station was studied in [23][24] whose goal was to maximize the local hit rate and data transmission rates while reducing service latency.

At the same time, it is an important avenue to solve network resource allocation problems by predicting network status. Content prefetching is widely used in mobile scenarios. Yuan et al. [25] used neural network to predict the users' requests information. Based on the predictive information, they designed a placement scheme to generate the placement strategy for edge node. Different from previous works, based on the assumption of synchronous offloading, Chen et al. [26] studied an energy-saving offloading strategy whose arrival time and task processing time are asynchronous. The problem was transformed into two subproblems. Authors combined the double DQN and distributed LMST to predict time intervals and reduce the overall solution's computational complexity. Cao et al. [27] conducted a study on the deployment of heterogeneous edge servers, whose goal is to minimize the expected response time of base station system. Based on the game theory, they designed a mobility-aware approach to analyze the movement of users. Although predicting network information can improve the effectiveness of decisions, the accuracy of prediction method is far from guaranteed. These works always depend on high prediction accuracy, which introduces certain limitations.

Focusing on computing services, an architecture which jointly optimizes computing and caching in 5G networks was proposed in [28]. In [29], the authors researched a joint computation offloading and data caching problem in a hybrid MEC system to minimize the request delay at the user side. However, these works all ignore the fairness in optimization, which is an important factor of user experience on the network. Few works on computing offloading considered fairness, but paper [30] is one of them. There are still several significant differences. First, [30] only focuses on the optimization of current time slot and ignores the future information, while we achieve the long-term average optimization. Second, [30] only considers the min-max fairness function, while we use the - fair utility function where min-max fairness is just one case. Third, we propose an innovate DRL algorithm to solve the problem and [30] is based on convex optimization theory.

III. System Modeling

This section introduces the system model for computing offloading, which includes system scenario, service delay model, and problem formulation. Table I lists the mathematical notations.

A. Scenario Description

The network scenario consists of cloud, base station (BS) and users, which is a collaborative cloud-edge network and illustrated in Fig.1. The V-antenna base station, whose storage space is C and computing capability (e.g. the maximum frequency of CPU) is F, works in full-duplex mode. It can cache service data (e.g. databases/libraries and content) and provide computation for services. However, BS can not support all services due to the limited resources.

The set of services is denoted as K , $f1; 2; :::; Kg$. For each service, there are three important attributes $(c_k; u_k; o_k)$, where c_k is the required storage space to cache service data, u_k is the required computation resource to finish computing, and o_k is the data size of output. Let's take live streaming as an example, c_k is the data size of related codec databases/libraries that application need to cache, u_k is total computation to encode and decode contents, and o_k is the data size of output codec streaming. Each independent service is assumed to be the smallest processing unit which is indivisible [32].

The time is slotted [33], i.e., $T = f1; 2; :::; Tg$. There are two phases in a time slot, user request phase and service providing phase. The beginning of a time slot is user request phase. In this phase, users generate service requests and send them to BS. We denote the number of users as N, and the number of requests for each service as a vector $D^t = [d_1^t; :::; d_k^t; :::; d_K^t]$, where d_k^t is the number of requests for service k in time slot t. For user n, we use $H_n^l = [h_{1;n}^l; h_{2;n}^l; :::; h_{V;n}^l]$ to denote the downlink channel matrix of BS, and w_n is the additive white Gaussian noise, whose covariance is $^2_{l;n}$

In the service providing phase, BS provides services. As mentioned above, BS cannot support all services locally because of the limited network resources of BS. We should make decisions based on the request status and caching status to determine which services can be processed locally for minimizing service delay. Then, BS will provide services based on the decision.

B. Service Delay Model

Local Computing: If BS provides service k locally, the computation delay is defined as:

$$tc_k^l = u_k / f_k^l \tag{1}$$

where f_k^l is the computation resources that the BS assigns to service k per second (cycles/s). If BS has cached related libraries and content, it can process the service directly. The received signal of user n, which consists of received radio

TABLE I: Mathematical Notations

Notation	Explanation	Notation	Explanation
F	Computing capacity of BS	C	Storage space of BS
K	Set of services	T	Set of time slots
u_k	Required computing resource of service k	c_k	Required storage space of service k
o_k	The data size of output	w_n	The additive white Gaussian noise
N	The number of users	M^t	Edge caching status at time slot t
D^t	The number of requests at time slot t	S^t	The state at time slot t
X^t	The action at time slot t	R^t	The reward at time slot t
H_n^l	The downlink channel matrix of BS	p_n^l	The power of transmission signal

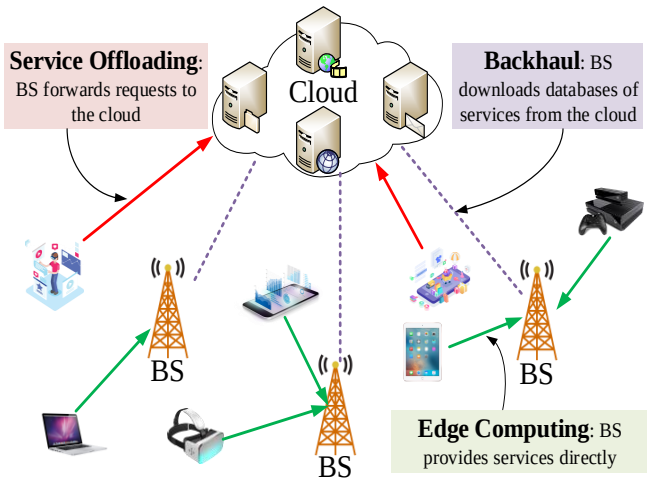


Fig. 1: System Architecture

frequency signal from BS and the additive white Gaussian noise at user n, can be expressed as:

$$y_n = \sqrt{p_n^l}(H_n^l)^T g_n^l + w_n \tag{2}$$

where $(\,)^T$ is transposition operation, g_n^l is transmission signal from BS to user n, and p_n^l is power of transmission signal. Then, we can calculate the signal to interference plus noise ratio (SINR) as:

$$\gamma_n^l = \frac{p_n^l tr f (H_n^l)^T (H_n^l)^{TH} g}{\sigma^2_{l;n}} \tag{3}$$

where $tr f g$ represents the trance of matrix, $(\,)^H$ is conjugate transpose of matrix, and p_n^l is transmission power of downlink that BS transmits signal to user n. The transmission rate is defined as:

$$r_n^l = b^l log_2(1 + \gamma_n^l) \tag{4}$$

where b^l is the channel bandwidth of BS. For user n, the transmission delay of service k is:

$$tr_k^l = o_k / r_n^l \tag{5}$$

The service delay consists of computation delay and transmission delay as following:

$$T_k^l = tc_k^l + tr_k^l \tag{6}$$

If BS does not cache related databases/libraries and content, it first needs to download related data from the cloud by backhaul, and the download time is $t_k^d = c_k / B$, where B is the bandwidth of wired link between SBS and the cloud. Therefore, the service delay in this case is:

$$T_k^l = tc_k^l + tr_k^l + t_k^d \tag{7}$$

Cloud Computing: If service k is processed in the cloud, the computation delay is:

$$tc_k^c = u_k / f_k^c \tag{8}$$

where f_k^c is the computation resources that the cloud assigns to service k per second. Due to the uncertainty of cloud transmission[34], we simplify the transmission rate as:

$$r^c = b^c log_2(1 + \frac{p^c h^c}{\sigma_c^2}) \tag{9}$$

where b^c, p^c, h^c, σ_c^2 respectively denote the channel bandwidth, fixed transmission power, the channel gain, noise power of cloud. The service delay is:

$$T_k^c = tc_k^c + tr_k^c \tag{10}$$

where $tr_k^c = o_k / r^c$ is the transmission delay of cloud computing.

C. Problem Formulation

Compared to cloud computing, edge computing can reduce service latency by supporting faster transmission. The saved time for service k by edge computing is calculated as:

$$T_k^s = T_k^c - T_k^l \tag{11}$$

The average saved time is:

$$\overline{T_k^s} = \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} (d_k^t x_k^t T_k^s) \tag{12}$$

where d_k^t is the number of requests, x_k^t is the computing status of service k, and $x_k^t = 1$ if BS provides service k locally, otherwise $x_k^t = 0$. At each time slot, the computing status of all services need to be determined, which can be denoted as a vector $X^t = [x_1^t; \ldots; x_k^t; \ldots; x_K^t]$.

In order to guarantee fairness, the α-fair utility function is introduced in our problem. For some constant $\alpha \geq 0$, the α-fair utility is defined as:

$$U(x) = \begin{cases} x^{1-\alpha}/(1-\alpha) & for\ \alpha \neq 1 \\ log(x) & for\ \alpha = 1 \end{cases} \tag{13}$$

The fairness-aware computing offloading is formulated as follows:

$$\max_{X^t} \sum_{k \in K} U(\overline{T_k^s})$$
$$s.t. \sum_{k \in K} c_k x_k^t \quad C \tag{14a}$$
$$\sum_{k \in K} f_k^l x_k^t \quad F \tag{14b}$$

where the goal is the -fair utility function of average reduced time, (14a) is the constraint of storage space due to the fact that BS needs to cache relevant databases if it provides services locally, (14b) is the computing capacity constraint of BS.

It is intractable to derive the optimal solution of above problem. First, we need to find the long-term average optimization solution of service computing problem. Traditional methods such as dynamic programming always need the complete state transition probability which is related to the user requests. These methods also make decisions only after knowing the user requests in all time slots, so they need to predict future information. However, this prediction is both difficult and impractical to make due to the dynamic characteristics of the network systems. Besides, the optimization problem (14) has huge solution space. In each time, we need to obtain the decision X^t, which has 2^K possibilities due to a combination explosion of options. It is clearly not feasible to traverse such huge solution space. Furthermore, there are many illegal actions because of the limited resource of BS, which increases the complexity of optimization. Third, reinforcement learning is good at solving long-term average optimization problems, but the goal of our problem is the -fair utility function of average reduced time not only the average reduced time, which further increases the difficulty of solving the problem.

IV. Markov Decision Process for Service Offloading

Reinforcement learning which replaces the state transition probability by data sampling can achieve model-free learning and is an efficient method to solve this kind of problem. It only needs current information and does not require to predict future data. Considering that deep reinforcement learning is an efficient method, we first formulate the computing offloading problem without fairness guarantee as an MDP. Three important elements are defined: state space, action space and reward function. Then, we adjust the reward for -fair utility function and analyze the stationary point of the reward-adjusted problem.

A. Markov Decision Process

We first formulate the original computing offloading problem without fairness guarantee, whose goal is long-term average reduced time, as following:

$$\max_{x_k^t} \sum_{k \in K} \overline{T_k^s}$$
$$s.t. (14a), (14b) \tag{15a}$$

Three important variables of MDP for the original problem are defined.

The State Space: In this problem, the system state consists of two variables, caching status and request status. We denote edge caching status as a vector $M^t = [m_1^t, \ldots, m_k^t, \ldots, m_K^t]$, where $m_k^t = 1$ if BS has cached the relevant databases of service k locally at the beginning of time slot t, otherwise $m_k^t = 0$. In order to describe the request status, we define a vector $D^t = [d_1^t, \ldots, d_k^t, \ldots, d_K^t]$ to represent the number of requests to each service, where d_k^t is the number of requests. Therefore, we denote the state at time slot t as $S^t = (M^t, D^t)$, and the state space is given as:

$$S = \{(M^t, D^t) | M^t \in M, D^t \in N, t \in T\} \tag{16}$$

where M is the set of caching status under the problem constraints, N is the set of all positive integers.

The Action Space: In the model, the action at each time slot is defined as the computing decision $X^t = [x_1^t, \ldots, x_k^t, \ldots, x_K^t]$. For action X^t, it consists of x_k^t which is called underlying action. The size of action space is 2^K, where K is the number of services. Besides, we call these actions which do not satisfy problem constraints as *illegal* actions.

The Reward Function: In this problem, an action can grant higher reward if it brings higher saved time. The reward function of original problem is divided into several components.

Illegal actions should be avoided, so we defined the reward associated with illegal actions to be Pu, which is a negative number, as a penalty. In legal actions, we discuss the reward depending on the saved time. If BS provides service k locally which means $x_k^t = 1$, the saved time by edge computing is $d_k^t(T_k^c \quad T_k^l)$. If the service k is completed in the cloud then the saved time is 0. The reward of underlying action x_k^t is summarized as follows:

$$r_k^t = \begin{cases} Pu; & if\ X^t\ is\ illegal \\ d_k^t(T_k^c \quad T_k^l); & if\ X^t\ is\ legal\ and\ x_k^t = 1 \\ 0; & if\ X^t\ is\ legal\ and\ x_k^t = 0 \end{cases} \tag{17}$$

where d_k^t is the request number of service k. By the way, the calculation of T_k^l is different for the different current caching status:

$$T_k^l = \begin{cases} tc_k^l + tr_k^l; & if\ m_k^t = 1 \\ tc_k^l + tr_k^l + t_k^d; & if\ m_k^t = 0 \end{cases} \tag{18}$$

The reward of action X^t is the sum of all reward of underlying action x_k^t:

$$R^t = \sum_{k \in K} r_k^t \tag{19}$$

B. Model Adjustment and Analysis

Our goal is to optimize the fairness utility $U(\overline{T_k^s})$ not the long-term average $\overline{T_k^s}$, so we need to adjust the MDP of original problem (15). It is easy to know that the past reward history of services affects the fairness of decision. We record the average of past reward until time slot t as:

$$h_k^t = \frac{1}{t} \sum_{=1} r_k \tag{20}$$

Then we use the average of past reward and original reward to get the adjusted reward, which is defined as follows:

$$\hat{r}_k^t = r_k^t U^0(h_k^t) = r_k^t U^0(\frac{1}{t}\sum_{=1} r_k) \quad (21)$$

where $U^0(x)$ is the first order derivative of the fairness utility function $U(x)$.

Theorem 1. *The stationary point of the problem which has adjusted reward is also a stationary point of the -fair utility optimization problem (14).*

Proof. Based on paper [35], we give the proof of **Theorem 1**. We use to denote a stationary point for the problem with adjusted reward, which means $r\hat{\rho} j_{=} = 0$, where $\hat{\rho}$ is the average adjusted reward under policy and is defined as:

$$\hat{\rho} = \lim_{T!1}\frac{1}{T}[\sum_{t=1}^{T}\sum_{k=1}^{K} r_k^t U^0(\sum_{=1} r_k = t)] \quad (22)$$

The average reward for service k of original problem Eq.(15) is as following:

$$p_{;k} = \lim_{T!1}\frac{1}{T}\sum_{t=1}^{T} r_k^t \quad (23)$$

The Markov Chain is irreducible and aperiodic under policy , so for any 0, we can get a T that satisfies the equation:

$$j\frac{1}{T}\sum_{t=1}^{T} r_k^t \quad p_{;k}j < \quad (24)$$

Combining Eq.(22), Eq.(23) and Eq.(24), we have:

$$\begin{aligned} & j\hat{\rho} \quad \sum_{k=1}^{K} p_{;k}U^0(p_{;k})j \\ = & \; j\frac{1}{T}\sum_{t=1}^{T}\sum_{k=1}^{K} r_k^t U^0(\frac{1}{t}\sum_{=1} r_k) \quad \sum_{k=1}^{K} p_{;k}U^0(p_{;k})j \\ & \sum_{k=1}^{K} j\frac{1}{T}\sum_{t=1}^{T} r_k^t U^0(\frac{1}{t}\sum_{=1} r_k) \quad p_{;k}U^0(p_{;k})j \\ & \sum_{k=1}^{K} j\frac{1}{T}\sum_{t=1}^{T} r_k^t U^0(p_{;k}) \quad p_{;k}U^0(p_{;k})j + \quad C_2 L \\ & \sum_{k=1}^{K} j\frac{1}{T}\sum_{t=1}^{T} r_k^t \quad p_{;k}j \quad jU^0(p_{;k})j + \quad C_2 L \\ & C_1 + \quad C_2 L \end{aligned} \quad (25)$$

where C_1 is a bound for $jU^0(p_{;k})j$ and C_2 is a bound for the average reward $\frac{1}{T}\sum_{t=1}^{T} r_k^t$. In the third step above, we used the Lipschitz continuity as following:

$$jU^0(\frac{1}{t}\sum_{=1} r_k) \quad U^0(p_{;k})j \quad Lj\frac{1}{t}\sum_{=1} r_k \quad p_{;k}j \quad L \quad (26)$$

As ! 0, that is T ! 1, we have $\hat{\rho} = \sum_{k=1}^{K} p_{;k}U^0(p_{;k})$. For -fair utility functions, we know that $(1 \quad)U(x) = xU^0(x)$ [1]. Therefore, $r\hat{\rho} j_{=} = 0$ implies following equation:

$$r[\sum_{k=1}^{K} p_{;k}U^0(p_{;k})]j_{=} = 0 \quad (27)$$

$$r[\sum_{k=1}^{K} U(p_{;k})]j_{=} = 0 \quad (28)$$

Therefore, the stationary point of adjusted-reward problem is also a stationary point for the -fair utility optimization problem (14). **Theorem 1** is proved.

From **Theorem 1**, we know that the -fair utility optimization problem and the adjusted reward problem have the same stationary point and we can use gradient policy to get it. This motivates us to design an algorithm to solve the problem.

V. Problem Solution

This section discusses the solution to the computing offloading problem with guarantee. We modify DQN in terms of adjusted reward, exploration strategy and update method. Then a novel DRL approach is proposed to solve the problem.

A. Proposed Solution Principle

Due to the lack of state transition probability matrix, traditional solutions such as policy iteration and value iteration fail to solve this problem. DRL is a more efficient way to solve long-term average optimization problem. It can learn the optimal strategy by data sampling without knowing the transition probability matrix.

As the MDP model described in previous section, we can simply apply a DRL method such as the DQN algorithm, whose input is state S^t and output is action X^t, to solve the service offloading problem. Although we can use DQN directly to solve our problem, there are two problems. First, as mentioned earlier, our optimization goal is the -fair utility function of average saved time not average saved time. DQN can not achieve this goal. Second, for each service, there are two options $x_k^t \in \{0, 1\}$. The size of action space reaches 2^K, which is exponential to the number of services. In DQN, we need an output neuron to represents the reward of an action, so the number of output neurons is very large and we require huge computing resources to train the model. Therefore, it is not wise to use the DQN algorithm directly and we need to reduce the action space.

For the first problem, we can use $\hat{r}_k^t$ to replace r_k^t as the analysis in Section IV-B. To deal with the second problem, we make more analysis. In the service offloading problem, equation (19) shows that the reward of an action X^t consists of its components x_k^t. The Q-values of all actions X^t can be calculated by the combinations of x_k^t if we have get the Q-value of underlying actions x_k^t. Therefore, we only need to train the expected reward of components x_k^t, rather than X^t. In this way, the action space can be reduced from 2^K to the

[1] When 1, we have $xU'(x) = x^{1-} = (1 \quad)U(x)$. When = 1, optimization of fair utility $log(x)$ can be considered as the limit of $(x^{1-} \quad 1)=(1 \quad)$ as ! 1

number of components $2K$ [2]. When $x_k^t = 0$, the saved time is 0 and the reward is 0, so we don't need to train. In other words, for service k, we only need to calculate the reward of providing it locally, which is also the reward of underlying action $x_k^t = 1$. Therefore, we further reduce the size of action space from $2K$ to the number of services K.

Although we have the idea of reducing action space, two important problems remain. First, the state transition depends on action X^t in this problem. But in our idea, the model should output the Q-value of underlying actions $x_k^t = 1$, which means we can not get action X^t and lead to the failure of state transition. Second, the loss function that we can get at each time slot is for action X^t. But we have to calculate the loss function of underlying action $x_k^t = 1$ to train the model, which is contradictory in traditional DRL algorithms.

Algorithm 1 Optimal Set Selection Algorithm

Require:
Storage space C;
Computing capacity F;
Attributes $(c_k; f_k^l)$ of all services;
Q-value of edge computing q_k
Ensure:
Computing status vector X^t;

```
for k = 1;:::;K do
  for j = 1;:::;C do
    for i = 1;:::;F do
      if c_k > j or f_k^l > i then
        V_{j;i}^k = V_{j;i}^{k-1}
        remove x_k^t if x_k^t in X^t
      else if V_{j-c_k;i-f_k^l}^{k-1} + q_k > V_{j;i}^{k-1} then
        V_{j;i}^k = V_{j-c_k;i-f_k^l}^{k-1} + q_k
        add x_k^t if x_k^t not in X^t
      else
        V_{j;i}^k = V_{j;i}^{k-1}
        remove x_k^t if x_k^t in X^t
      end if
    end for
  end for
end for
```

To solve above problems, we modify the classical DQN algorithm from two aspects. The first aspect is *exploration strategy*. The output of our model is the Q-value of underlying actions $x_k^t = 1$. In the classical DQN algorithm, agent can only select an underlying action by -greedy policy each time. However, the state transition of the model is based on action X^t not underlying action $x_k^t = 1$, which means the exploration strategy should obtain a set of actions rather than only an action. Therefore, a new selection mechanism of optimal actions set whose Q-value is largest is proposed, which will be described in detail in subsection V-C. The new exploration strategy of our model is that we select a random actions set with a probability of , otherwise we get the optimal

[2]There are two options for each service and the number of services is K. So the number of components is product of the two $2K$.

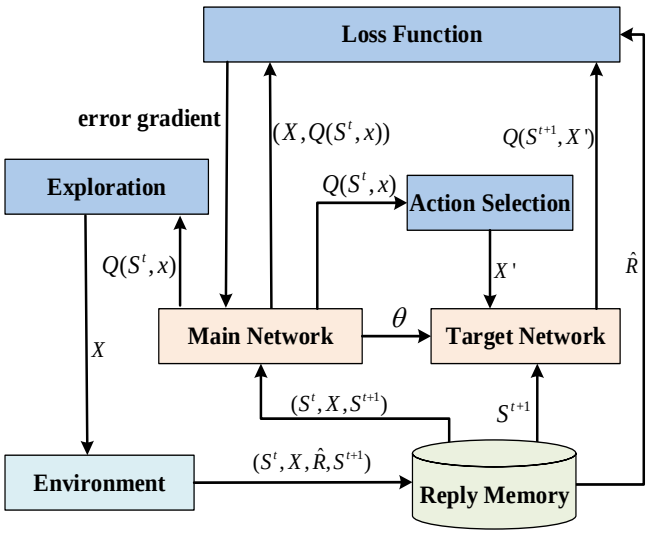


Fig. 2: Process of Solution

actions set by our new selection mechanism with a probability of 1 .

The second innovative aspect is the *update of Q-values*. In our model, we need to calculate the loss function of underlying action x_k and update model. But we only can get the reward of action X^t from system. This is contradictory because action X^t is a set of underlying actions and we cannot use its reward to calculate the loss function of underlying action x_k directly. So, we first calculate the loss function of action X^t:

$$L(X^t) = (Q_r^0 \quad Q(S^t; X^t))^2 \tag{29}$$

where $Q_r^0 = R + \quad Q^0(S^{t+1}; argmax_{X^0} Q(S^{t+1}; X^0))$, X^0 is the optimal underlying actions set that selected by our new selection mechanism, and $Q(S^t; X^t)$ is the Q-value of set X^t. The calculation of $Q(S^t; X^t)$ is as follows:

$$Q(S^t; X^t) = \sum_{x_k^t 2 X^t} Q(S^t; x_k^t) \tag{30}$$

Then, the loss function of underlying action x_k^t in action X^t is:

$$L(x_k^t) = \frac{Q(S^t; x_k^t)}{Q(S^t; X^t)} \quad L(X^t) \tag{31}$$

Finally, we can use the loss function $L(x_k^t)$ for $8x_k^t \; 2 \; X^t$ to update our main network model.

In MURL algorithm, we use the loss value of action X to calculate the loss value of underlying actions x which belong to action X. So, in each episode, we get the loss value of multiple underlying actions x and update parameters of the neural network model multiple times. This is unlike in traditional DRL, where the parameters can only be updated once in each episode. As the MURL algorithm updates the values of an action set, rather than updating the value of one action only in each episode, it was denoted *multi-update*.

B. Optimal Set Selection Mechanism

In this subsection, we will introduce the optimal set selection mechanism in detail, which can get the action set with

largest Q-value. We first formulate the problem as follows:

$$\begin{aligned} \max \quad & \sum_{x_k \in X} Q(S, x_k) \\ s.t. \quad & \sum_{x_k \in X} c_k \quad C \\ & \sum_{x_k \in X} f_k^l \quad F \end{aligned} \tag{32}$$

The goal of problem is to optimize the Q-value, and the constraints are the limited storage space and computing capacity. This is a multi-constraint 0-1 Knapsack problem with profits $Q(S, x_k)$ and weights $C; F$. If the storage space of BS is j and the computing capability is i, we denote $V_{j,i}^k$ as the maximum Q-value of optimal action set for first k services. The transition equation (i.e. recursion) of $V_{j,i}^k$ is:

$$V_{j,i}^k = \begin{cases} V_{j,i}^{k-1}, & if \quad c_k > j \quad or \quad f_k^l > i \\ \max\{V_{j,i}^{k-1}, V_{j-c_k,i-f_k^l}^{k-1} + q_k\}, & otherwise \end{cases} \tag{33}$$

where c_k represents the storage space to cache the service data, the f_k^l represents the computation resource that the SBS provides for service k, and the q_k is the Q-value of action x_k.

For equation (33), service k cannot be supported locally if its required storage space or computation resource exceeds the local resources of base station. In this case, the optimal action set with largest Q-value for the first k services is equal to first $k-1$ services. If BS has enough resources to support service k, there are two options. If BS processes service k locally, the total Q-value is $V_{j-c_k,i-f_k^l}^{k-1} + q_k$. If service k is completed in cloud, the total Q-value is the same as the first $k-1$ services $V_{j,i}^{k-1}$. This process is introduced in **Algorithm 1**.

C. Proposed Service Offloading Algorithm

Fig.2 illustrates the architecture of our algorithm. To overcome the problem of overestimation, we use two multi-layer neural networks with the same structure: *Main Network* and *Target Network*. In the training, we need to train the parameters of main network by loss function, and use target network to get the target Q-values. The target network copies the main network parameters at regular intervals. As mentioned, we use Q-value of underlying action $Q(S^t, x_k^t)$ instead of Q-value of underlying actions set $Q(S^t, X^t)$ to reduce the action space from 2^K to K. So, the output of our models are the Q-value of underlying action $Q(S^t, x_k^t)$, whose size is K. In the state transition of model, we need underlying action set $Q(S^t, X^t)$ to determine the next state. The *Action Selection* module can achieve this transition by obtaining the optimal underlying set according to **Algorithm 1**. In the update of model parameters, we need the loss function of underlying action x^t. *Loss Function* module can calculate it based on the reward of underlying action set X. The *Exploration* module selects a proper action, as mentioned in subsection V-A.

The training of model needs computing resources and is time consuming. As BS computing resources are often limited, to reduce the pressure on BS, we consider cooperation between BS and the cloud to train the model. The main idea is as follows. BS transmits the experience information $(S^t, X^t, \hat{R}^t, S^{t+1})$ to the cloud. The cloud uses this experience information to train the model and transmits the model parameters back to the BS. The proposed MURL algorithm is described in **Algorithm 2** and **Algorithm 3**.

Algorithm 2 MURL Algorithm in BS

Require:
Model Parameters ;
Ensure:
Four-tuple of experience information;
1: **for** $t = 1, \ldots, T$ **do**
2: Sent request to the cloud for model parameters;
3: Recieve the model parameters from the cloud;
4: Get current state S^t;
5: Select action X^t by new exploration strategy;
6: Execute action X^t,get next state S^{t+1} and reward R^t;
7: Calculate adjusted reward $\hat{b}_k^t$ by Eq.(21);
8: Calculate $\hat{R}^t$;
9: Sent four-tuple $(S^t, X^t, \hat{R}^t, S^{t+1})$ to the cloud;
10: **end for**

Algorithm 2 indicates the process of information collection performed at BS.

The cloud trains the model. The training process is introduced by **Algorithm 3**.

In MURL, there are several improvements in comparison with DQN. In line 5 of **Algorithm 2**, we use our new exploration strategy to get a set of underlying actions X instead of a single underlying action x_k. In line 8 of **Algorithm 2**, we use the adjusted reward to optimize the -fair utility function. Additionally, the reward $\hat{R}^j$ is for set X^j, so we cannot directly use the Q-value of the underlying action x_k^j to calculate the loss function. We have to get the Q-value of set X^j by employing **Algorithm 1**, as shown in lines 11 and 12 of **Algorithm 3**. Lines 13-18 of **Algorithm 3** correspond to the update process. We calculate the loss function of set X^j and get the loss function of each underlying action in X^j by proportional distribution. Finally, for all underlying actions in X^j, we update the main network according to the loss function. As users' preferences change slowly in time, the model does not need a real-time update. For example, the model can be updated and trained once every so often (e.g. two weeks) and in between, we can use the already trained model to make decisions.

It is worth noting that there are two major aspects which strongly recommend using multi-update deep reinforcement learning to solve the proposed problem. First, it is the problem's huge and discrete action space. The action space size of the computing offloading problem is 2^K. We use MURL to train the Q-value of underlying action and reduce the action space from 2^K to K. To solve any conflicts in training, MURL employs a new exploration strategy and an improved Q-value update method. Second, it is the optimization objective of the problem. Our goal is optimizing the -fair utility function of the long-term average saved time and not the long-term average saved time. In MURL, we record the average of past rewards, then use this average of past rewards and the original reward to get the adjusted reward. We also analyze

Algorithm 3 MURL Algorithm in the cloud

Require:
Learning rate ; Decay factor ;
The update frequency of target network e;
Ensure:
The main network parameters ;
Initialize the main network parameter and Q-values $Q(S; x_k)$ randomly;
Initialize target network parameter $^0 =$ and Q-value $Q^0(S; x_k) = Q(S; x_k)$;
Initialize S^1 as first state and get S^1;
for $i = 1; :::; P$ **do**
  if receive the parameters request from BS **then**
    send main network parameters to BS
  end if
  if receive four-tuple $(S^t; X^t; R^t; S^{t+1})$ **then**
    Store $(S^t; X^t; R^t; S^{t+1})$ in replay memory;
    Sample m samples $(S^j; X^j; R^j; S^{j+1})$ from replay memory randomly;
    Select $argmax_{X^0} Q(S^{j+1}; X^0)$ by **Algorithm 1** in main network;
    Calculate $Q^0(S^{j+1}; X^0)$ by (30) in target network
    Compute the target value for action set X^j

$$y^j = \begin{cases} R^j; & \text{terminate} \\ R^j + Q^0(S^{j+1}; X^0); & \text{otherwise} \end{cases}$$

    Get the loss function $L(X^j)$ by equation (29) ;
    Get $L(x_k)$ by equation (31) for $8x_k \ 2 \ X^j$
    for $8x_k \ 2 \ X^j$ **do**
      Perform gradient descent with respect to the network parameters by $L(x_k)$;
    end for
    if $i\%e == 0$ **then**
      Update target network parameter $^0 =$
    end if
  end if
end for

the stationary point of the reward-adjusted problem.

VI. Simulation Results

We design extensive simulation base on TensorFlow [36] to demonstrate the performance of our algorithm as efficient solution to the service offloading problem. In this section, we will show the simulation results and analyze the performance in terms of convergence, fairness, computing capacity, storage space, and user requests. Table II summarizes the experimental settings.

A. Simulation Scenario and Setup

There are 10 independent services. For service k, the range of required computing resource u_k is $[0.5GHz; 2.5GHz]$, the range of required storage space c_k is $[1.2GB; 2GB]$ and the data size of output range is $[4MB; 6MB]$. The bandwidth B of backhaul from cloud to BS is set to $8Gbps$ and the download delay can be calculated as $_k = 8 \ c_k=B$. Similar to [37], we assume that the request frequency conforms the Zipf distribution with parameters $= 1$ and $V = 0.1$. The average number of requests to service k is as follows:

$$D(k) = \frac{V}{r(k)} \quad N \tag{34}$$

where $r(k)$ is the ranking of user request frequency, N is the number of users. In each time, the user request number follows the Poisson distribution whose parameter is $D(k)$. The default computing capacity of BS is $10GHz$, and the default storage space is $10GB$. In terms of data transmission, the transmission rate from BS to user is between $[80Mbps; 100Mbps]$ and the transmission rate from cloud to user is between $[8Mbps; 15Mbps]$. The transmission time is calculated as $tc = 8 \ o_k=r$, where o_k is the data size of output and r is the transmission rate.

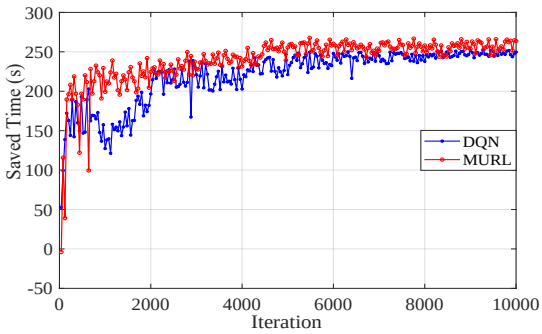
TABLE II: Parameters setting for simulations

Parameters	Value
The number of services	10
The number of users	300
Computing capacity of BS	$10GHz$
Storage space of BS	$10GB$
Required computing resource	$[0.5GHz; 2.5GHz]$
Required storage space	$[1.2GB; 2GB]$
Bandwidth of backhaul	$8Gbps$
Data size of output	$[4MB; 6MB]$
Transmission rate from BS to users	$[80Mbps; 100Mbps]$
Transmission rate from cloud to users	$[8Mbps; 15Mbps]$
Batch size	32
Learning rate	0.01
Decay factor	0.9

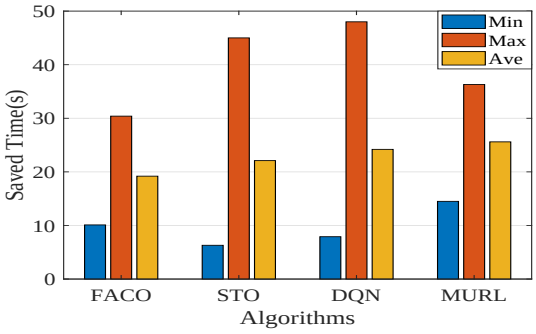
The neural network employed has three layers. The first layer is the input layer, which is responsible for taking the state as input and passing the data to the following layers. The number of neurons in the input layer is $2K$, where K is the number of services. In the experimental setup considered, the input layer has 20 neurons. The second layer is a fully-connected (FC) layer and there are 20 neurons with rectified linear units (ReLU). The last layer is the output layer. The number of neurons in the output layer is K, (i.e. set to 10 in experiment). In the model training phase, we set the batch size to 32, the learning rate is 0.01 and the decay factor is 0.9.

Due to the issues related to interpretability of neural networks and randomness of training data, it is intractable to prove the gap between our proposed algorithm and the corresponding theoretical optimal solution [38]. Additionally, the long-term average performance optimization problem is NP-hard. Therefore very few approaches get the optimal solutions of such a problem by employing variations of traditional optimization algorithms (e.g. convex optimization) and many of them use suboptimal solutions [27]. Therefore, for the evaluation, we compare the performance of our algorithm with that of three other baseline solutions via simulation-based experiments. Table III summarizes the differences between our algorithm and three baselines.

Single Time slot Optimization(STO) This solution [3] designs an asymmetric search tree and improves the branch and bound method to solve the computing offloading problem. STO focuses on optimizing current system



(a)



(b)

Fig. 3: Performance of algorithms (a) convergence behavior; (b) fairness

TABLE III: Comparison of algorithms

Algorithm	Fairness	Optimization	Method
STO	No	Single time slot	Asymmetric search
DQN-based	No	Long-term average	DQN
FACO	Yes	Single time slot	Convex optimization
MURL	Yes	Long-term average	A novel DRL algorithm

performance and not the long-term average performance and does not take the fairness into account.

DQN-based solution: This solution [39] is to optimizes the long-term average performance of edge computing system based on DQN algorithm. But it doesn't consider the fairness of services and the constraint of storage space. To compare with our algorithm, we add the constraint of storage space in the experiment.

Fairness-awared Computing Offloading solution (FACO): This solution [30] considers the network optimization of current time slot and use convex optimization theory to solve this problem. It also takes the fairness into account.

B. Comparative Performance Evaluation

This section discusses the performance of our algorithm in terms of convergence and service delay.

Fig.3(a) shows the convergence behaviors in the training phase. We only show the performance of MURL and DQN-based solution, because FACO and STO solutions are not based on reinforcement learning and doesn't need the training phase. In MURL, after 4600 training episodes, BS establishes the knowledge model of the whole network system. The average of saved time is stable and the algorithm converges. The DQN-based solution converges after 8200 episodes. It can also be seen that MURL outperforms the DQN-based solution, as MURL updates Q-values for an actions set in each episode, while DQN-based solution performs update only once per action. Fig.3(b) shows the performance of four algorithms in fairness. We can find that MURL has the largest average saved time, but the gap between maximum saved time and minimum saved time is small. This indicates that our algorithm MURL has good performance in service delay and fairness. Although the average saved time of DQN-based solution is larger than FACO, the gap between maximum saved time and minimum saved time of FACO is smaller than DQN-based solution. The reason is that FACO only considers the optimization for current time slot not for the long-term average, which reduces the average saved time. Although DQN-based solution has larger average saved time, it ignores the fairness. As a result, DQN-based solution has the largest gap between maximum saved time and minimum saved time. Similarly, STO considers the performance optimization for the current time slot only, and ignores fairness, so it saves more time, but it is very unfair.

Fig.4 shows the saved time by employing edge computing during testing phase. The saved time is defined as the service delay of cloud computing minus the delay of edge computing, which consists of reduced transmission and download delay. By the way, the reduced transmission time is defined as the difference of transmission time between edge computing and cloud computing which is calculated as $tc_k^c + tr_k^c \quad tc_k^l \quad tr_k^l$. Download delay t_k^d is the increased download delay for BS downloading related databases from cloud. We set the initial state of BS to *null*, that is, BS does not cache any services databases. In Fig.4(a), the reduced transmission time of MURL is similar to DQN-based solution, because it will sacrifice part of the service delay to ensure the fairness of users. As Fig.4(a) shows, MURL and DQN-based solution, which have been trained with historical data during the training phase, have faster convergence speed than FACO and STO. FACO and STO only consider the optimization for the current time slot, so their time reduction is less than those of DQN and MURL which consider long-term average optimizations. Fig.4(b) shows the download delay of the four solutions. At the beginning, FACO and STO solution frequently download services as there is a lack of historical data. This is also illustrated in Fig.4(a). Due to the off-line training of the model in advance, MURL and DQN-based can adapt to the environment quickly and have low download delay. Besides, we can find that the frequency of services download is much smaller than the frequency of service requests, which also shows that caching is performed on a much larger time-scale [40]. Fig.4 shows that MURL has the larget saved time and the best system performance in the four solutions.

C. Impact of Computing Capacity

We analyzes the performance of four solutions under the constraint of computing capacity in this subsection. Fig.5 shows the effect of computing capacity on saved time and

(a) (b)

Fig. 4: Saved time (a) reduced transmission time; (b) delay for downloading databases

(a) (b)

Fig. 5: Effect of computing capacity (a) saved time; (b) backhaul traf c

backhaul traf c. We adjusted the BS computing capacity to test, and other parameters are consistent with Table II.

Fig.5(a) shows the impact of computing capacity on saved time when the storage space is xed at 10GB. The saved time of four solutions all increases with the computing capacity of BS. It is obvious that BS can process more services locally if its computing capacity increases. Due to the limited of storage space, we cannot always increase the saved time by increasing computing capacity of BS, and the effect of computing capacity gradually weakens until disappears. In addition, different algorithms have different utilization of computing capacity, resulting in different stationary points. According to the average results of multiple experiments, our algorithm saves up to 24.6% more time compared to FACO, 11.2% more than when STO is used, and 4.3% more than the DQN-based solution. The reason is that FACO and STO only optimize the single time slot performance and often get a local optimal solution. Our algorithm and DQN-based solution optimize long-term average performance, which leads to better performance, in favor of MURL.

Fig.5(b) illustrates the impact of computing capacity on backhaul traf c. The backhaul traf c is de ned as the load rate of backhaul link. It decreases with the increase of computing capacity, contrary to saved time. Because more services can be provided by BS when computing capacity increases, and BS forwards less requests to cloud. Therefore, the services provided by cloud decrease, which results in the decrease of backhaul traf c. Besides, the backhaul traf c of MURL algorithms is the minimum, DQN is the second best, STO is the third, and FACO has the maximum value, which means MURL has the best performance in terms of edge of oading, followed by the other algorithms in the indicated order.

D. Impact of Storage Space

This subsection analyzes the impact of storage space on the performance, shown in Fig.6. We adjusted the storage space to test, and the other parameters are consistent with Table II. Fig.6(a) shows the impact of storage space on saved time when the computing capacity is xed at 10GHz. Similar to the situation of computing capacity, lager storage space can also increase the saved time and has bottleneck. One different is that the saved time of FACO solution doesn't change with the storage space and always stays the same in picture, because the limited computing capacity of BS cannot support more services under the strategy of FACO.

Fig.6(b) illustrates the backhaul traf c decreases with the increase of storage space, and the trend levels off. The reason is that more storage space means more service database can be stored in BS which decreases the backhaul traf c. But the backhaul traf c tends to be stable because of the constraint of computing capacity. According to Fig.6(b), our algorithm MURL reduces the backhaul traf c with up to 16.1%, 8.7% and 5.6% compared to FACO, STO and DQN-based solution, respectively.

From Fig.5 and Fig.6, we can nd that both storage space and computing capacity can affect service quality, but the constraints are coupled. If we only change one aspect, it often has bottleneck for the overall system performance improvement.

(a) (b)

Fig. 6: Effect of storage space (a) saved time; (b) backhaul traf c

(a) (b)

Fig. 7: Effect of number of contents (a) number of users; (b) parameters of Zipf distribution

E. Impact of User Request

Both the number of users and Zipf distribution parameter can affect the number of user requests. These two factors are studied in the following experiment.

The effect of number of users is shown in Fig.7(a). We set the Zipf distribution parameter as $= 1$. From equations (17) and (34), we know that the saved time is proportional to the number of requests and the number of requests is proportional to the number of users. So, the saved time of the four algorithms is proportional to number of users, which corresponds to the data illustrated in the picture.

Fig.7(b) shows how the Zipf distribution parameter affects the performance of our solution MURL. There are $N = 300$ users in experiment. We nd that the saved time is inversely proportional to Zipf distribution parameter. Because larger parameter mean less requests according to equation (34), so the total saved time decreases. To intuitively re ect the change of each user, we divide the total saved time by the number of users and get the average saved time. The right y-axis of Fig.7(b) shows that the average saved time increases with Zipf distribution parameter, and they are positively correlated. Zipf distribution parameter becomes large means the user requests are more concentrated to services with high popularity rankings. Providing these popular services at edge nodes can improve more system performance. Take an extreme example, if Zipf distribution parameter goes to 0 and all services have the same frequency of requests, all algorithms will get closer to the random-based solution without considering the constraints.

VII. Conclusions

This paper focused on the long-term average performance optimization of the cloud-edge computing of oading problem with fairness guarantee. First, we formulated the problem with the goal of minimizing the save time. This problem cannot be solved with traditional methods due to the lack of state transition probability. To provide a solution, we introduced a novel DRL algorithm and designed the Markov decision process of the problem. We adjusted the reward function and proved the stationary point theoretically to guarantee the fairness of services. Due to the huge solution space, we improved the traditional DQN algorithm from exploration strategy and model update, and then proposed MURL, a new algorithm which can effectively reduce the size of the action space from 2^K to K. Extensive simulations show that proposed solution outperforms three alternative approaches in terms of different indicators including convergence, fairness and service delay. Future work will focus on the computation resources allocation problem in a dynamic network with a cloud and multiple BSs.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (NSFC) via grant 62225105; the Natural Natural Science Foundation of Shandong Province under Grant. ZR2022QF040; the QLU Pilot Project of Integration of Science, Education and Production under Grant. 2022PX083 and 2022GH007. G.-M. Muntean acknowledges the support of the Science Foundation Ireland (SFI) grants 21/FFP-P/10244 (Faradis) and 12/RC/2289_P2 (Insight).

REFERENCES

[1] Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," IEEE Commun. Surveys Tuts., vol. 19, no. 4, pp. 2322-2358, 2017.

[2] X. Chen, W. Li, S. Lu, Z. Zhou and X. Fu, "Efficient Resource Allocation for On-Demand Mobile-Edge Cloud Computing," IEEE Trans. Veh. Technol., vol. 67, no. 9, pp. 8769-8780, Sept. 2018.

[3] J. Zhang et al., "Joint Resource Allocation for Latency-Sensitive Services over Mobile Edge Computing Networks with Caching," IEEE Internet Things J., vol. 6, no. 3, pp. 4283-4294, June 2019.

[4] J. Wang, J. Hu, G. Min, A. Y. Zomaya and N. Georgalas, "Fast Adaptive Task Offloading in Edge Computing Based on Meta Reinforcement Learning," IEEE Trans. Parallel and Distributed Syst., vol. 32, no. 1, pp. 242-253, Jan. 2021.

[5] Y. Li, X. Wang, X. Gan, H. Jin, L. Fu and X. Wang, "Learning-Aided Computation Offloading for Trusted Collaborative Mobile Edge Computing," IEEE Trans. Mobile Comput., vol. 19, no. 12, pp. 2833-2849, 2020.

[6] M. Chen, A. Liu, W. Liu, K. Ota, M. Dong and N. N. Xiong, "RDRL: A Recurrent Deep Reinforcement Learning Scheme for Dynamic Spectrum Access in Reconfigurable Wireless Networks," IEEE Trans. Netw. Sci. Eng., vol. 9, no. 2, pp. 364-376, 1 March-April 2022.

[7] S. Guo, J. Liu, Y. Yang, B. Xiao and Z. Li, "Energy-Efficient Dynamic Computation Offloading and Cooperative Task Scheduling in Mobile Cloud Computing," IEEE Trans. Mobile Comput., vol. 18, no. 2, pp. 319-333, Feb 2019.

[8] H. Wang, Z. Lin and T. Lv, "Energy and Delay Minimization of Partial Computing Offloading for D2D-Assisted MEC Systems," in Proc. IEEE Wireless Communications and Networking Conference (WCNC), 2021.

[9] 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., early access.

[10] A. Moubayed, A. Shami, P. Heidari, A. Larabi and R. Brunner, "Edge-enabled V2X Service Placement for Intelligent Transportation Systems," IEEE Trans. Mobile Comput., vol.20, no.4, pp.1380-1392, April 2021.

[11] Z. Jiang, C. Xu, J. Guan, Y. Liu and G. Muntean, "Stochastic Analysis of DASH-Based Video Service in High-Speed Railway Networks," IEEE Trans. Multimedia, vol. 21, no. 6, pp. 1577-1592, June 2019.

[12] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," IEEE J. Sel. Areas Commun., vol. 34, no. 12, pp. 3590–3605, 2016.

[13] S. David et al., "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, no. 7587, pp. 484-489, 2016.

[14] H. Hao, C. Xu, L. Zhong and G. Muntean, "A Multi-update Deep Reinforcement Learning Algorithm for Edge Computing Service Offloading," in Proc. ACM Multimedia (ACM MM) 2020.

[15] X. Chen et al., "Augmented Queue-Based Transmission and Transcoding Optimization for Livecast Services Based on Cloud-Edge-Crowd Integration," IEEE Trans. Circuits Syst. Video Technol., vol. 31, no. 11, pp. 4470-4484, Nov. 2021.

[16] X. Nie, L. T. Yang, J. Feng and S. Zhang, "Differentially Private Tensor Train Decomposition in Edge-Cloud Computing for SDN-based Internet of Things," IEEE Internet Things J., vol. 7, no. 7, pp. 5695-5705, 2020.

[17] Y. Zhou, F. R. Yu, J. Chen and Y. Kuo, "Communications, Caching, and Computing for Next Generation HetNets," IEEE Wireless Commun., vol. 25, no. 4, pp. 104-111, Aug. 2018.

[18] P. Ren, X. Qiao, Y. Huang, L. Liu, S. Dustdar and J. Chen, "Edge-Assisted Distributed DNN Collaborative Computing Approach for Mobile Web Augmented Reality in 5G Networks," IEEE Netw, vol. 34, no. 2, pp. 254-261, March 2020.

[19] D. T. Hoang, D. Niyato, D. N. Nguyen, E. Dutkiewicz, P. Wang and Z. Han, "A Dynamic Edge Caching Framework for Mobile 5G Networks," IEEE Wireless Commun., vol. 25, no. 5, pp. 95-103, Oct. 2018.

[20] Y. Guan, X. Zhang and Z. Guo, "PrefCache: Edge Cache Admission With User Preference Learning for Video Content Distribution," IEEE Trans. Circuits Syst. Video Technol., vol. 31, no. 4, pp. 1618-1631, April. 2021.

[21] M. Chen, W. Liu, A. Liu, Z. Zeng, "Edge intelligence computing for mobile augmented reality with deep reinforcement learning approach," Comput. Networks, vol. 195, 2021.

[22] J. Dai, Z. Zhang, S. Mao and D. Liu, "A View Synthesis-Based 360° VR Caching System Over MEC-Enabled C-RAN," IEEE Trans. Circuits Syst. Video Technol., vol. 30, no. 10, pp. 3843-3855, Oct. 2020.

[23] X. Zhao, P. Yuan, H. li and S. Tang, "Collaborative Edge Caching in Context-Aware Device-to-Device Networks," IEEE Trans. Veh. Technol., vol. 67, no. 10, pp. 9583-9596, Oct. 2018.

[24] J. Kwak, Y. Kim, L. B. Le and S. Chong, "Hybrid Content Caching in 5G Wireless Networks: Cloud Versus Edge Caching," IEEE Trans. Wireless Commun., vol. 17, no. 5, pp. 3030-3045, May 2018.

[25] X. Yuan, M. Sun and W. Lou, "A Dynamic Deep-learning-based Virtual Edge Node Placement Scheme for Edge Cloud Systems in Mobile Environment," IEEE Trans. Cloud Comput., vol. 10, no. 2, pp. 1317-1328, 2022,

[26] M. Chen, W. Liu, T. Wang, S. Zhang and A. Liu, "A game-based deep reinforcement learning approach for energy-efficient computation in MEC systems," Knowl. Based Syst., vol. 235, 2022.

[27] K. Cao, L. Li, Y. Cui, T. Wei and S. Hu, "Exploring Placement of Heterogeneous Edge Servers for Response Time Minimization in Mobile Edge-Cloud Computing," IEEE Trans. Indus. Inform., vol. 17, no. 1, pp. 494-503, 2021.

[28] K. Zhang, S. Leng, Y. He, S. Maharjan and Y. Zhang, "Cooperative Content Caching in 5G Networks with Mobile Edge Computing," IEEE Wireless Commun., vol. 25, no. 3, pp. 80-87, June 2018.

[29] X. Yang, Z. Fei, J. Zheng, N. Zhang and A. Anpalagan, "Joint Multi-User Computation Offloading and Data Caching for Hybrid Mobile Cloud/Edge Computing," IEEE Trans. Veh. Technol., vol. 68, no. 11, pp. 11018-11030, Nov. 2019.

[30] J. Du, L. Zhao, J. Feng and X. Chu, "Computation Offloading and Resource Allocation in Mixed Fog/Cloud Computing Systems With Min-Max Fairness Guarantee," IEEE Trans. Commun., vol. 66, no. 4, pp. 1594-1608, April 2018.

[31] X. Wang, R. Li, C. Wang, X. Li, T. Taleb and V. C. M. Leung, "Attention-Weighted Federated Deep Reinforcement Learning for Device-to-Device Assisted Heterogeneous Collaborative Edge Caching," IEEE J. Sel. Areas Commun., vol. 39, no. 1, pp. 154-169, Jan. 2021.

[32] L. Wei, C. H. Foh, B. He and J. Cai, "Towards Efficient Resource Allocation for Heterogeneous Workloads in IaaS Clouds," IEEE Trans. Cloud Comput., vol. 6, no. 1, pp. 264-275, Jan. 2018.

[33] S. Li et al, "Joint Admission Control and Resource Allocation in Edge Computing for Internet of Things," IEEE Netw, vol. 32, no. 1, pp. 72-79, Feb. 2018.

[34] X. Hu, L. Wang, K. -K. Wong, M. Tao, Y. Zhang and Z. Zheng, "Edge and Central Cloud Computing: A Perfect Pairing for High Energy Efficiency and Low-Latency," IEEE Wireless Commun., vol. 19, no. 2, pp. 1070-1083, Feb. 2020.

[35] J. Chen, Y. Wang and T. Lan, "Bringing Fairness to Actor-Critic Reinforcement Learning for Network Utility Optimization," Proc. IEEE Conference on Computer Communications(IEEE INFOCOM), pp. 1-10, 2021.

[36] A. Martin, A. Ashish, B. paul and B. Eugene, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems" 2015. https://cse.buffalo.edu/ chandola/teaching/mlseminardocs/TensorFlow.pdf.

[37] M. Zhang, H. Luo and H. Zhang, "A Survey of Caching Mechanisms in Information-Centric Networking," IEEE Commun. Surveys Tuts., vol. 17, no. 3, pp. 1473-1499, 2015.

[38] E. Weinan, et al., "Towards a Mathematical Understanding of Neural Network-Based Machine Learning: what we know and what we don't," CSIAM Trans. Applied Mathematics, 2020.

[39] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji and M. Bennis "Optimized Computation Offloading Performance in Virtual Edge Computing Systems Via Deep Reinforcement Learning," IEEE Internet Things J., vol. 6, no.3, pp. 4005-4018, 2019.

[40] A. F. Molisch et al., "Caching eliminates the wireless bottleneck in video aware wireless networks," Adv. Elect. Eng. pp 1-13, 2014.