IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) | 978-1-6654-0926-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/INFOCOMWKSHPS54753.2022.9797916

# DDPG-based Computation Offloading and Service Caching in Mobile Edge Computing

Lingxiao Chen*, Guoqiang Gong*, Kai Jiang†, Huan Zhou*, and Rui Chen*
*College of Computer and Information Technology, China Three Gorges University, Yichang, China
†School of Cyber Science and Engineering, Wuhan University, Wuhan, China

***Abstract*—Mobile Edge Computing (MEC) migrates the computing center to the network edge to provide computing services for Mobile Users (MUs). However, due to the limited capacity of MUs and insufficient types of services stored by edge servers, it brings great challenges to computation offloading and service caching in Internet of Things (IoT) network. In this paper, we investigate the joint optimization problem of computation offloading, resource allocation and service caching replacement in a collaborative MEC system to minimize the total cost of all MUs under the latency and resource constraints. Accordingly, we formulate the problem as a Mixed Integer Non-linear Programming (MINLP) problem. In order to solve this problem, the Deep Deterministic Policy Gradient (DDPG)-based algorithm is proposed. The simulation results show that the proposed algorithm is better than other benchmark schemes in different scenarios.**

***Index Terms*—mobile edge computing, computation offloading, resource allocation, service caching, DDPG.**

## I. Introduction

With the rapid development of 5G technique, intelligent terminals have brought great convenience to people's lives. Nevertheless, it cannot meet mobile applications such as low latency and high reliability because of the limited functions of intelligent terminals, which greatly affects the user experience. In this case, Mobile Users (MUs) can offload local tasks to the cloud by exploiting Mobile Cloud Computing (MCC). However, cloud servers usually have high transmission latency because of long distance, which are not suitable for latency-sensitive applications. Against this background, Mobile Edge Computing (MEC) is emerging as a new means to deal with the above problems and is widely used in various communication networks, such as Internet of Things (IoT) [1]–[3].

In recent years, a range of literatures have focused on computation offloading in IoT networks [4]–[7]. In [8], the authors have jointly considered computation offloading, content caching, and resource allocation as an integrated model to minimize the total latency consumption of the computation tasks. In [9], the authors have proposed the joint computation offloading and resource allocation problem as Markov Decision Process (MDP) to intelligently minimize the system energy consumption. In [10], the authors have investigated a weighted multi-objective optimization problem to minimize the total offloading energy consumption and latency. In [11], the authors have proposed a consolidated stochastic computation offloading framework to address the growing computational requirements of MEC-based Industrial IoT (IIoT) systems. In [12], the authors have considered a multi-user noncooperative computation offloading game to adjust the offloading probability of each vehicle in vehicular edge computing networks.

In addition, caching is also a significant factor that affects computation offloading of the tasks, which mainly includes two types. The first type is content caching. In this case, frequently-used contents are cached in the edge server. In [13], the authors have proposed a content caching strategy based on edge caching user classification to maximize the cache hit probability. In [14], considering that the instantaneous content popularity may vary over time, the authors have designed a dynamic probability cache scheme that can predict the average content prevalence over a time window. The second type is service caching. In this case, the edge service caching refers to the program that is stored on an edge server to perform computing tasks. The authors have combined communication, caching and computing to reduce the total latency of the system [15]. In [16], the cooperative caching schemes to reduce network latency and increase network throughput have been proposed. In [17], the authors have presented an intelligent computation offloading system with service caching, which aims to minimize task processing time and long-term energy utilization.

Nevertheless, the above studies seldom consider the cooperation among Base Stations (BSs). Meanwhile, due to the limitation of computing resources and caching capacity, the BS equipped an edge server cannot meet the requirements of all computation offloading tasks. Therefore, it is of great significance to design a suitable computation offloading scheme to ensure the full utilization of edge servers' computing resources and caching capacity in IoT network. Motivated by this, in this paper, the joint optimization problem of computation offloading and service caching as well as edge collaboration in multi-user and multi-edge server MEC system is considered. Then, a Deep Deterministic Policy Gradient (DDPG) -based algorithm is proposed to solve the optimization problem. Simulation results show that the proposed DDPG-based algorithm greatly outperforms other benchmark algorithms in different scenarios.

The rest of this paper is organized as follows. Section II introduces the main assumptions, system model as well as problem formulation. Section III proposes the DDPG-based


978-1-6654-0926-1/22/$31.00 ©2022 IEEE

algorithm. Section IV presents the simulation results. Finally, conclusions are summarized in Section V.

## II. SYSTEM MODEL

In this section, we introduce the system model and the main assumptions, as well as problem formulation.

### A. Network Architecture and Main Assumptions

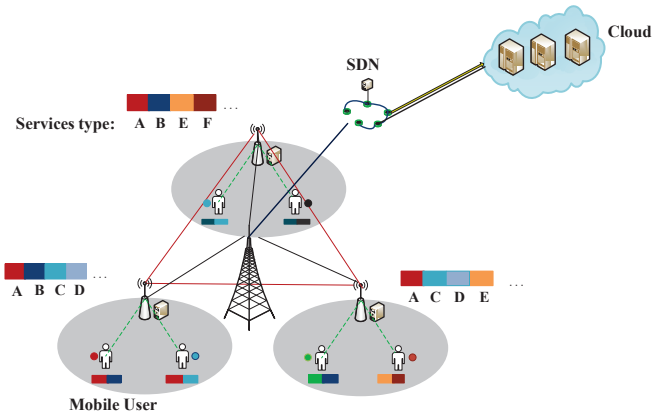


Fig. 1. Network Architecture

As shown in Fig. 1, we consider a IoT network, which consists of a cloud, a Software Defined Network (SDN), $N$ BSs as well as $I$ MUs. In our model, each BS is equipped with an edge server, which provides computing resources for MUs, and caches some services used to process computing tasks. SDN is used to collect and update the global status information continuously, such as Channel Status Information (CSI), service types cached at BSs, etc. It is assumed that each MU, say $\text{MU}_i$ $(i \in 1, 2, ..., I)$, generates a task and needs to pay relevant fees when offloading the tasks, which includes traffic fees and service fees. Meanwhile, $\text{MU}_i$ can offload the computing tasks to the local $\text{BS}_j$ $(j \in 1, 2, \ldots, N)$ and cooperative $\text{BS}_k$ $(k \in 1, 2, \ldots, N, k \neq j)$ or the cloud, where all service types are cached in the cloud.

Assumed that, the system operates within a fixed slot length $t = \{1, 2, \ldots, T\}$, where $T$ denotes the finite time horizon. In each time slot, $\text{MU}_i$ generates a computing task type $m$ $(m \in 1, 2, \ldots, M)$, and the task can be denoted as $Q_{i,m} \triangleq (C_{i,m}, D_{i,m}, T_{i,m}^{max})$, $C_{i,m}$ denotes the total number of CPU cycles required to complete the task, $D_{i,m}$ indicates the data size of the task, and $T_{i,m}^{max}$ is the latency constraint of the task. Meanwhile, the storage space required for the task type $m$ can be defined as $h_m$.

In our network, $\text{MU}_i$ transmits the service request to SDN with the aid of the BS. Then, SDN generates an optimal scheme by exploiting the current information. We assume that the transmission latency of the request can be ignored because of small size. Meanwhile, if the task cannot be performed by the local $\text{BS}_j$, it may be transferred to the cooperative $\text{BS}_k$ or the cloud with the aid of local $\text{BS}_j$.

Moreover, it is assumed that transmission interference can be ignored by exploiting Orthogonal Frequency Division Multiple Access (OFDMA) technique. Then, in the time slot $t$ , the achievable rate from $\text{MU}_i$ to $\text{BS}_j$ is, given by

$$R_{j,t} = B_i \log_2(1 + \frac{P_i g_{i,j}}{\sigma_j^2}), \tag{1}$$

where $B_i$ and $P_i$ are the bandwidth and transmission power of $\text{MU}_i$, respectively. $g_{i,j}$ denotes the channel gain between $\text{MU}_i$ and $\text{BS}_j$, $\sigma_j^2$ is the local noise at $\text{BS}_j$.

In addition, it is assumed that the BSs and cloud are connected through broadband, the transmission rate form $\text{BS}_j$ to $\text{BS}_k$ and cloud can be defined as $R_{k,t}$ and $R_{c,t}$, respectively.

### B. Latency

From the above, the offloading latency consists of transmission latency and computing latency. Let us first introduce transmission latency. In our model, when $\text{MU}_i$ transmits the offloaded task, the latency from $\text{MU}_i$ to $\text{BS}_j$, $\text{BS}_k$ and the cloud are, respectively, given by

$$T_{i,j,t}^{\text{Trans}} = \frac{D_{i,m}}{R_{j,t}}, \tag{2}$$

$$T_{i,k,t}^{\text{Trans}} = \frac{D_{i,m}}{R_{k,t}} + \frac{D_{i,m}}{R_{j,t}}, \tag{3}$$

$$T_{i,c,t}^{\text{Trans}} = \frac{D_{i,m}}{R_{c,t}} + \frac{D_{i,m}}{R_{j,t}}, \tag{4}$$

According to the offloading decision of SDN, $\text{MU}_i$ can offload tasks to the local equipment, the local BS, cooperative BS or cloud. Then, the computing latency is described as follow.

1) **Local Computing**: In this case, the task is performed by $\text{MU}_i$ and the latency is given by

$$T_{i,t}^{\text{Comp}} = \frac{C_{i,m}}{F_i}, \tag{5}$$

where $F_i$ is the computing resource of $\text{MU}_i$.

2) **Offloading to local $\text{BS}_j$**: In this case, $\text{MU}_i$ offloads the task to local $\text{BS}_j$. Then, the latency can be expressed as

$$T_{i,j,t}^{\text{Comp}} = \frac{C_{i,m}}{F_{i,j}}, \tag{6}$$

where $F_{i,j}$ denotes the computing resources allocated to the task by $\text{BS}_j$.

3) **Offloading to cooperative $\text{BS}_k$**: In this case, the task is implemented by $\text{BS}_k$ when the service is not cached at $\text{BS}_j$ or the computing resources of $\text{BS}_j$ are insufficient. Then, the latency is given by

$$T_{i,k,t}^{\text{Comp}} = \frac{C_{i,m}}{F_{i,k}}, \tag{7}$$

where $F_{i,k}$ denotes the computing resources allocated to the task by $\text{BS}_k$.

4) **Offloading to the cloud**: In this case, the service is not cached on any BS and the task is offloaded to the cloud. It is

assumed that the computing latency can be ignored because of the powerful resources of the cloud.

Hence, in the time slot $t$, the total execution latency can be expressed as

$$T_{i,t}^{\text{Total}} = \begin{cases} T_{i,t}^{\text{Comp}}, & a_{i,j,t}=0 \\ T_{i,j,t}^{\text{Trans}} + T_{i,j,t}^{\text{Comp}}, & a_{i,j,t}=1 \\ T_{i,k,t}^{\text{Trans}} + T_{i,k,t}^{\text{Comp}}, & a_{i,k,t}=1, k \neq j \\ T_{i,c,t}^{\text{Trans}}, & a_{i,c,t}=1 \end{cases} \tag{8}$$

where $a_{i,j,t} = 0$ represents that the task is executed on $\text{MU}_i$, and $a_{i,j,t} = 1$ represents that the task is offloaded to local $\text{BS}_j$. $a_{i,k,t} = 1$ $(k \neq j)$ denotes that the task is offloaded to cooperative $\text{BS}_k$. In addition, $a_{i,c,t} = 1$ indicates that the task is offloaded to the cloud.

### C. Overhead Model

As can be seen from the above, different offloading situations produce different payment expenses, which mainly include communication overhead caused by data transmission and service overhead caused by computing resources. Therefore, the two cases are considered. Firstly, when the task is performed by the cloud, the communication and service overhead are given by

$$O_{i,c,t} = W_c F_{i,c} + D_{i,m} V_j, \quad a_{i,c,t} = 1 \tag{9}$$

where $W_c$ and $V_j$ are the unit price of computing resources and communication, respectively, and $F_{i,c}$ is the computing resources assigned to this task by the cloud.

Then, there are two cases when the task is offloaded to the BS. One is offloaded directly to $\text{BS}_j$ and the other to $\text{BS}_k$. Hence, the overhead caused by the local $\text{BS}_j$, given by

$$O_{i,j,t} = W_j F_{i,j}, \quad a_{i,j,t} = 1 \tag{10}$$

where $W_j$ is the computing resource unit price of $\text{BS}_j$. In addtion, the overhead caused by the cooperative $\text{BS}_k$ can be given by

$$O_{i,k,t} = D_{i,m} V_j + W_k F_{i,k}, \quad a_{i,k,t} = 1, k \neq j \tag{11}$$

where $W_k$ is the computing resource unit price of $\text{BS}_k$.

### D. Problem Formulation

Therefore, the total cost of all MUs are the weighted sum of the latency and the overhead, given by

$$\begin{aligned} Cost_t^{\text{Total}} = & \alpha \Big( \sum_{i \in Set_{c,t}} O_{i,c,t} + \sum_{j \in N} \sum_{i \in Set_{j,t}} O_{i,j,t} + \\ & \sum_{k \in N} \sum_{i \in Set_{k,t}} O_{i,k,t} \Big) + (1-\alpha) \sum_{i \in I} T_{i,t}^{\text{Total}}, \end{aligned} \tag{12}$$

where $\alpha$ denotes the preference proportion of $\text{MU}_i$ for the overhead. $Set_{c,t}$ , $Set_{j,t}$ and $Set_{k,t}$ are the set of MUs that offload the tasks to the cloud, local $\text{BS}_j$ and cooperative $\text{BS}_k$, respectively.

In this paper, we investigate the joint optimization problem of computation offloading and resource allocation as well as service caching strategy to minimize the total cost, which can be written as

$$\min \lim_{T \to \infty} \quad \frac{1}{T} \sum_{t=1}^{T} Cost_t^{\text{Total}} \tag{13}$$

$$s.t. \quad a_{i,j,t} = \{0,1\}, \tag{13a}$$

$$s_{j,m,t} = \{0,1\}, \tag{13b}$$

$$a_{i,k,t} = \{0,1\}, \tag{13c}$$

$$a_{i,c,t} = \{0,1\}, \tag{13d}$$

$$\sum_{m=1}^{M} s_{j,m,t} h_m \leq U_j, \tag{13e}$$

$$T_{i,t}^{\text{Total}} \leq T_{i,m}^{\max}, \tag{13f}$$

$$0 \leq \sum_{i \in Set_j} F_{i,j} \leq F_j. \tag{13g}$$

where (13e) indicates that the total size of cached services should not exceed the storage capacity of edge servers, denoted by $U_j$; (13g) indicates that the computing resources allocated to $\text{MU}_i$ are not greater than the available computing resources, denoted by $F_j$. In our model, the environment is assumed to change dynamically. In order to find the optimal computation offloading, resource allocation and service caching strategy, the system needs to collect global information and make the optimal decision. Moreover, the objective function is a Mixed Integer Non-linear Programming (MINLP) problem, which becomes more complicated as the number of MUs increases. Therefore, we propose a DDPG-based algorithm to address this problem in dynamic and complex scenario.

## III. PROBLEM SOLVING

In this section, in order to minimize the cost of all MUs under latency and services type constraint, we utilize a DDPG-based algorithm to solve the optimization problem. Firstly, let us introduce the concepts about state function, action function as well as reward.

### A. Problem Formulation based on DDPG

In our model, tasks are randomly generated and offloaded to the BS. The BS can handle these tasks by itself or assisted by the cooperative BS or even the cloud, and finally return the results to MUs. Hence, the interaction process of $\text{MU}_i$ and the BS as well as the cloud can be seen as a MDP, which can be denoted by the tuple $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma\}$, $\mathcal{S}$ represents the state space, $\mathcal{A}$ represents the action space, $\mathcal{P}$ represents the state transition probability, $\mathcal{R}$ is the reward and $\gamma$ is discount factor. Therefore, the key elements about MDP can be defined as follows.

**State Space**: The state in MDP specifically refers to the state space that reflects network environment. $\mathcal{F}_t = \{F_{1,t}, F_{2,t}, \ldots, F_{N,t}\}$ is the resources of BSs at the current time slot $t$, $\mathcal{U}_t = \{U_{1,t}, U_{2,t}, \ldots, U_{N,t}\}$ is the the caching capacities of BSs at the current time slot $t$ and $\mathcal{Z}_t = \{s_{j,m,t} \mid j \in N, m \in M\}$ is the service caching placement state of BSs at time slot $t$. Therefore, the state space can be defined as $s_t = \{\mathcal{F}_t, \mathcal{U}_t, \mathcal{Z}_t\}$.

**Action Space**: The action space consists of there parts, the computation offloading strategy $\mathcal{X}_t = \{a_{i,j,t}, a_{i,k,t}, a_{i,c,t} \mid i \in I, j, k \in N\}$, the computing resource allocation strategy $\mathcal{Y}_t = \{f_{i,j,t} \mid i \in I, j \in N\}$, and the service caching placement strategy of BSs $\mathcal{Z}_t = \{s_{j,m,t} \mid j \in N, m \in M\}$. Therefore, the action space can be expressed as $a_t = \{\mathcal{X}_t, \mathcal{Y}_t, \mathcal{Z}_t\}$.

**Reward**: The goal is to minimize the cost of all MUs, which is contrary to achieve the largest cumulative discounted reward of DDPG. Therefore, according to the value of the objective function, the reward function can be defined as $r_t = -Cost_t^{\text{Total}}$.

### *B. Proposed DDPG-based Algorithm*

It is known that traditional reinforcement learning is limited to a small action space and sample space, which is generally discrete. Considering our model, DDPG-based Algorithm is utilized to solve the proposed optimization problem. According to the architecture of actor-critic, we can divide the DDPG-based algorithm into four parts: namely actor current network, actor target network, critic current network and critic target network. Next, we define the parameters of networks: $\theta^{\mu}$ and $\theta^{\mu'}$ are the parameters of actor current network and actor target network, respectively; $\theta^{Q}$ and $\theta^{Q'}$ are the parameters of critic current network and critic target network, respectively.

Then, DDPG-based algorithm can be expressed as follows. At the beginning, the centralized agent collects information from the environment to form a state space. After obtaining the initial state $s_t$, we input the current state into the actor current network to obtain the action $a_t$. In order to enable the agent to explore potential better strategies, the Uhlenbeck-Ornstein random process and random noise are utilized to train the network. Further, the next action $s_{t+1}$ and reward $r_t$ can be obtained after executing the selected action. Meanwhile, the tuple $\{s_t, a_t, r_t, s_{t+1}\}$ is stored in the replay buffer to update network parameters by randomly sampling a tuple. According to the current state and action, the current Q value value and the target Q value can be calculated by the critic network to minimize the loss function. Next, the actor current network utilizes the policy gradient to update the current policy. Then, the parameters of the target network are updated.

From the above, the action $a_t$ can be expressed as

$$a_t = \mu\left(s_t \mid \theta^{\mu}\right) + \mathcal{N}_t, \tag{14}$$

where $\mathcal{N}_t$ is random noise to enable the agent to explore potential better strategies. After randomly selecting data $\{s_i, a_i, r_i, s_{i+1}\}$ from the experience replay memory and transferring it to the actor and critic networks, the target Q value is given by

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} \mid \theta^{\mu'}) \mid \theta^{Q'}), \tag{15}$$

Here, network parameters $\theta^{Q}$ are updated in temporal-difference (TD) error mode, and the loss function can be expressed as

$$L = \frac{1}{N}\sum_{i}(y_i - Q(s_i, a_i \mid \theta^{Q})^2), \tag{16}$$

**Algorithm 1** DDPG-based joint Computation Offloading, Resource Allocation and Service Caching algorithm.

**Input:** state space $S$, action space $A$, the parameters about the testbed environments;

**Output:** Optimal action selection for computation offloading, resource allocation and service caching;

Initialize the actor and critic network parameters: $\theta^{Q}, \theta^{\mu}$;
Initialize the target actor and critic network parameters: $\theta^{Q'}, \theta^{\mu'}$;
Initialize replay buffer $\xi$;
**for** each episode **do**
  Randomly generate an initial state;
  **for** each step of episode **do**
    Choose an action with current strategy and noise: $a_t = \mu\left(s_t \mid \theta^{\mu}\right) + \mathcal{N}_t$;
    Import action $a_t$ in a custom environment, obtain total cost as reward and a new state $s_{t+1}$;
    Store transition $\{s_t, a_t, r_t, s_{t+1}\}$ in $\xi$;
    Randomly select data $\{s_i, a_i, r_i, s_{i+1}\}$ from the repaly buffer $\xi$ and transfer it to the actor and critic networks;
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} \mid \theta^{\mu'})$;
    Update the critic network by minimizing loss function by the equation (16);
    Update the actor network by using the gradient ascent by the equation (17);
    Update the critic target network and the actor target network parameters:
    $\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}$;
    $\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$;
  **end for**
**end for**

Then, the actor current network utilizes the policy gradient to update the current policy with the aid of $\theta^{Q}$ and sample tuples, given by

$$\nabla_{\theta^{\mu}}\mu|_{s_i} \approx \frac{1}{N}\sum_{i} \nabla_a Q(s, a|\theta^{Q})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}}\mu(s|\theta^{\mu})|_{s_i}, \tag{17}$$

In addition, unlike Deep Q Network (DQN), DDPG utilizes soft update to update parameters to improve learning stability. Hence, the update process of $\theta^{Q'}$ and $\theta^{\mu'}$ can be expressed as

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'}, \tag{18}$$

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}. \tag{19}$$

The details of DDPG-based algorithm can be summarized by Algorithm 1.

## IV. NUMERICAL RESULTS

In this section, we provide a comprehensive performance evaluation on the proposed algorithm. Without loss of generality, the main assumptions utilized in our simulations are summarized as follows: $B_i$=20MHz, $\sigma_j^2 = 10^{(-13)}$W, $g_{i,j}$=$127+30\cdot\log(L)$ [18], $L$ is the distance between MUs and BSs. Since the caching capacity of the MUs equipments is less

TABLE I
SIMULATION PARAMETERS

| Parameters | Value |
|---|---|
| Number of BSs | 4 |
| Number of MUs | 8 |
| Number of task types | 10 |
| BSs coverage radius | 200 m |
| Computing capacity of MU | 0.8 GHz |
| Computing capacity of BS | 10 GHz |
| Caching capacity of BS | [10, 60] GB |
| Each service storage requirement | [2, 6] GB |

than that of the edge server, it is reasonable to set the number of service types that the equipment of $MU_i$ can cache from 1 to 3. Then, the computing capacity of MUs equipments and the edge server are set to 0.8 GHz and 10 GHz, respectively. The caching capacity of the edge server is in the range of [10, 60] GB. Then, the simulation parameters of our system are shown in Table I.

To evaluate the proposed DDPG-based algorithm, it is compared with AC algorithm as well as other two benchmark schemes as follows: All Local (AL) scheme considers the computation tasks of all MUs to be performed locally; All Offloading (AO) scheme considers that all computing tasks are offloaded to the corresponding BS without any cooperation and resource optimization.

In Fig. 2, we compare the convergence of the DDPG-based and AC algorithms. It can be seen that the curves of the two algorithms have relatively serious jitter in the initial stage. This is due to the randomness of the early reinforcement learning process. As the number of episodes increases, the DDPG-based algorithm and AC algorithm converge to a stable value at about 1000 episodes and 1500 episodes, respectively. Therefore, we can draw a conclusion that DDPG-based algorithm has better performance than AC algorithm.

Fig. 3 shows the total cost versus the number of MUs. Apparently, the DDPG-based algorithm performs best, followed by the AC algorithm. In addition, it can be seen that the total cost of AO scheme is rapidly increase than other methods when the number of MUs increases. It is due to the fact that the increase of the number of MU leads to a shortage of computing resources, and the computing resources allocated to each MU are insufficient. Meanwhile, it can be seen that the proposed algorithm has the lowest total cost compared to other schemes.

Fig. 4 shows the total cost versus different computing capacity of the edge server. As can be seen from the figure, the DDPG-based algorithm gives the lowest total cost, followed by the AC algorithm. Considering the characteristics of the AL scheme, it dose not utilize the resource of BSs, and the value remains unchanged. Meanwhile, the total cost of all methods except AL scheme has decreased when the computing capacity of the edge server increases. That's because as the computing resources of the edge server increase, each MU can allocate more available computing resources, and the computing latency is significantly reduced. Hence, it effectively reduces the total cost.

Fig. 5 shows the relationship between the caching capacity

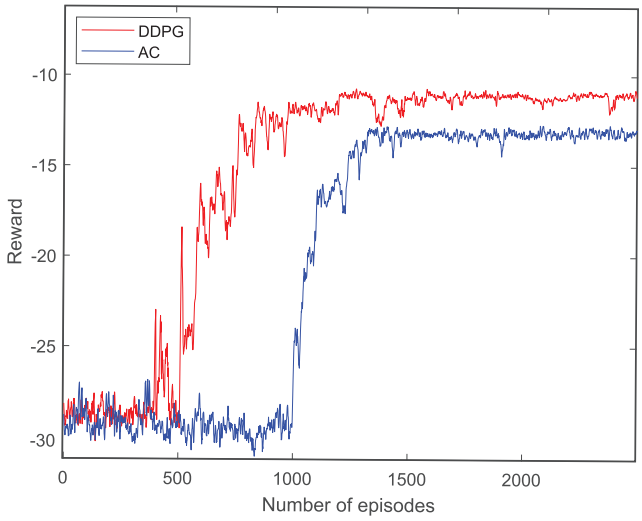


Fig. 2. The convergence of the proposed DDPG-based algorithm and AC algorithm.

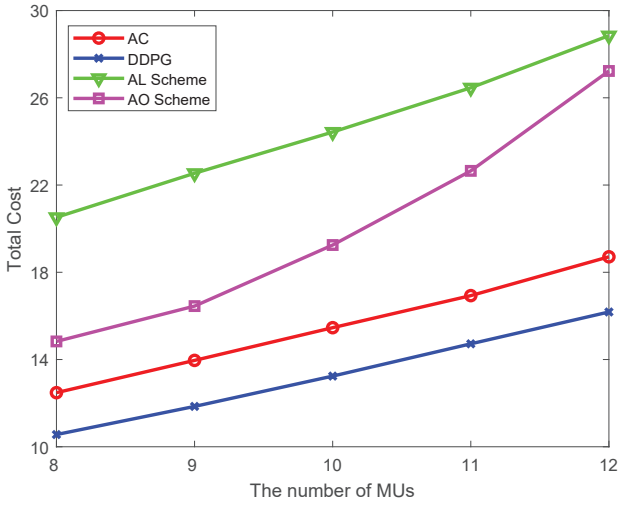


Fig. 3. Total cost versus the number of MUs.

of the edge server and total cost. Obviously, it can be seen that the total cost generated by the DDPG-based algorithm is the lowest among the four methods. It is due to the fact that the DDPG-based method optimizes the decisions of computation offloading, resource allocation, service caching replacement and edge collaborating at the same time. Thus, the purpose of minimizing MUs' cost can be achieved. The increase of edge caching capacity means that more applications can be cached in the edge server to provide computing services for MUs. Otherwise, whether they are executed on the equipment side or request from the cloud can lead to higher cost. In addition, it can be found that the curve tends to flatten when the caching capacity of the edge server reaches 40 GB, indicating that the main factor affecting MUs' cost is no longer the caching capacity, but the computing resources.

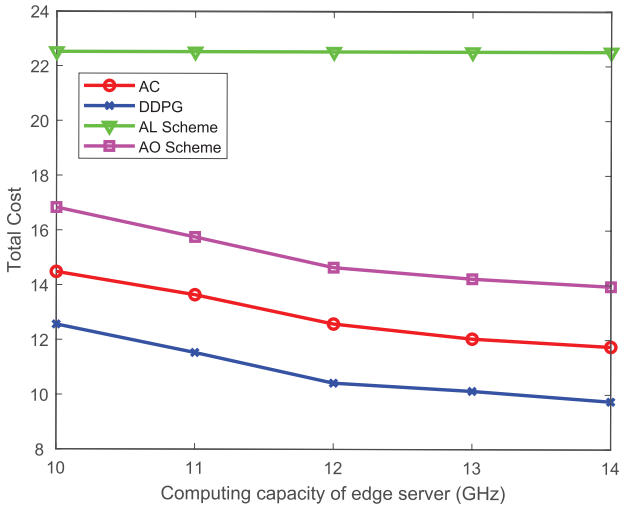


Fig. 4. Total cost versus the computing capacity of edge server

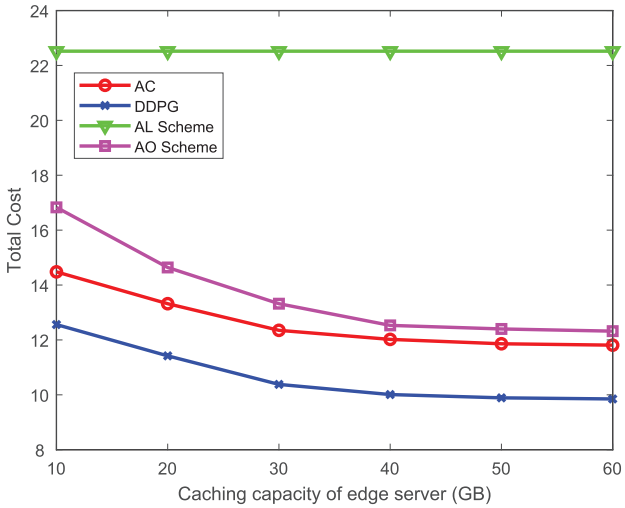


Fig. 5. Total cost versus the caching capacity of edge server

In summary, the computing resources and caching capacity of edge server has a remarkable influence on the experimental results, and the proposed DDPG-based algorithm outperforms other benchmark methods in terms of the long-term cost of all MUs in different scenarios, which validates the effectiveness of our proposed method.

## V. CONCLUSION

In this paper, we have investigated the joint optimization of computation offloading, resource allocation and service caching in multi-user and multi-edge server MEC system. In order to enhance the service quality of MUs, we have comprehensively considered the latency and the overhead. Moreover, the DDPG-based algorithm is utilized to minimize the total cost of all MUs, which obtains the optimal strategy of computation offloading, resource allocation and service caching. The simulation results show that the proposed algorithm outperforms the benchmark schemes. While this paper has focused on the single agent and binary offloading in our model, a natural extension is to investigate the multi-agent and partial offloading in future work.

## ACKNOWLEDGEMENT

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants NO. 62172255 and NO. 61872221. The corresponding author is H. Zhou (zhouhuan117@gmail.com).

## REFERENCES

[1] G. Premsankar, M. Di Francesco, and T. Taleb, "Edge computing for the internet of things: A case study," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1275–1284, 2018.

[2] K. Jiang, C. Sun, H. Zhou, X. Li, M. Dong, and V. C. Leung, "Intelligence-empowered mobile edge computing: Framework, issues, implementation, and outlook," *IEEE Network*, vol. 35, no. 5, pp. 74–82, 2021.

[3] Y. Qu, H. Dai, H. Wang, C. Dong, F. Wu, S. Guo, and Q. Wu, "Service provisioning for uav-enabled mobile edge computing," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 11, pp. 3287–3305, 2021.

[4] Y. Qu, D. Lu, H. Dai, H. Tan, S. Tang, F. Wu, and C. Dong, "Resilient service provisioning for edge computing," *IEEE Internet of Things J.*, vol. PP, no. 99, pp. 1–1, 2021.

[5] Y. Qu, H. Dai, F. Wu, D. Lu, C. Dong, S. Tang, and G. Chen, "Robust offloading scheduling for mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. PP, no. 99, pp. 1–1, 2020.

[6] B. Brik, P. A. Frangoudis, and A. Ksentini, "Service-oriented mec applications placement in a federated edge cloud architecture," in *Proc. IEEE ICC*, 2020, pp. 1–6.

[7] H. Zhou, T. Wu, H. Zhang, and J. Wu, "Incentive-driven deep reinforcement learning for content caching and d2d offloading," *IEEE J. Sel. Areas Commun*, vol. 39, no. 8, pp. 2445–2460, 2021.

[8] Y. Dai, D. Xu, S. Maharjan, G. Qiao, and Y. Zhang, "Artificial intelligence empowered edge computing and caching for internet of vehicles," *IEEE Wirel. Commun.*, vol. 26, no. 3, pp. 12–18, 2019.

[9] M. T. Kabir, M. R. Khandaker, and C. Masouros, "Minimizing energy and latency in fd mec through multi-objective optimization," in *Proc. IEEE WCNC*, 2019, pp. 1–6.

[10] S. Bebortta, D. Senapati, C. R. Panigrahi, and B. Pati, "An adaptive performance modeling framework for qos-aware offloading in mec-based iiot systems," *IEEE Internet Things J.*, vol. PP, no. 99, pp. 1–1, 2021.

[11] Y. Wang, P. Lang, D. Tian, J. Zhou, X. Duan, Y. Cao, and D. Zhao, "A game-based computation offloading method in vehicular multiaccess edge computing networks," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4987–4996, 2020.

[12] C. Ma, M. Ding, H. Chen, Z. Lin, G. Mao, Y.-C. Liang, and B. Vucetic, "Socially aware caching strategy in device-to-device communication networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4615–4629, 2018.

[13] Z. Sang, S. Guo, and Y. Wang, "Collaborative video cache management strategy in mobile edge computing," in *Proc. IEEE WCNC*, 2021, pp. 1–6.

[14] K. Guo and T. Q. Quek, "On the asynchrony of computation offloading in multi-user mec systems," *IEEE Trans. Commun.*, vol. 68, no. 12, pp. 7746–7761, 2020.

[15] H. Hua and X. Chu, "Content caching policy with edge caching user classification in fog radio access networks," in *Proc. IEEE WCNC*, 2021, pp. 1–7.

[16] J. Gao, S. Zhang, L. Zhao, and X. Shen, "The design of dynamic probabilistic caching with time-varying content popularity," *IEEE. Trans. Mob. Comput.*, vol. 20, no. 4, pp. 1672–1684, 2020.

[17] G. Zheng, C. Xu, H. Long, and X. Zhao, "Mec in noma-hetnets: A joint task offloading and resource allocation approach," in *Proc. IEEE WCNC*, 2021, pp. 1–6.

[18] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, 2018.