Hindawi
Wireless Communications and Mobile Computing
Volume 2020, Article ID 2765491, 17 pages
https://doi.org/10.1155/2020/2765491

Research Article

Deep Reinforcement Learning for Performance-Aware Adaptive Resource Allocation in Mobile Edge Computing

Binbin Huang,[1] Zhongjin Li,[1] Yunqiu Xu,[2] Linxuan Pan,[3] Shangguang Wang,[4] Haiyang Hu,[1] and Victor Chang[5]

[1]*School of Computer, Hangzhou Dianzi University, Hangzhou, China 310018*
[2]*School of Software, University of Technology Sydney, Ultimo, NSW 2007, Australia*
[3]*State Key Laboratory for Novel Software Technology, Software Institute, Nanjing University, Nanjing, China*
[4]*State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China*
[5]*School of Computing, Engineering and Digital Technologies, Teesside University, Middlesbrough, UK*

Correspondence should be addressed to Zhongjin Li; lizhongjin@hdu.edu.cn

Received 4 December 2019; Revised 10 February 2020; Accepted 15 June 2020; Published 2 July 2020

Academic Editor: Massimo Condoluci

Mobile edge computing (MEC) enables to provide relatively rich computing resources in close proximity to mobile users, which enables resource-limited mobile devices to offload workloads to nearby edge servers, and thereby greatly reducing the processing delay of various mobile applications and the energy consumption of mobile devices. Despite its advantages, when a large number of mobile users simultaneously offloads their computation tasks to an edge server, due to the limited computation and communication resources of edge server, inefficiency resource allocation will not make full use of the limited resource and cause waste of resource, resulting in low system performance (the weighted sum of the number of processed tasks, the number of punished tasks, and the number of dropped tasks). Therefore, it is a challenging problem to effectively allocate the computing and communication resources to multiple mobile users. To cope with this problem, we propose a performance-aware resource allocation (PARA) scheme, the goal of which is to maximize the long-term system performance. More specifically, we first build the multiuser resource allocation architecture for computing workloads and transmitting result data to mobile devices. Then, we formulate the multiuser resource allocation problem as a Markova Decision Process (MDP). To achieve this problem, a performance-aware resource allocation (PARA) scheme based on a deep deterministic policy gradient (DDPG) is adopted to derive optimal resource allocation policy. Finally, extensive simulation experiments demonstrate the effectiveness of the PARA scheme.

1. Introduction

In recent years, smart mobile devices have become an indispensable part of people's lives. With the popularity of smart mobile devices (e.g., smartwatches, smart glasses, and smartphones), various types of mobile applications, such as virtual reality (VR), augmented reality (AR), and interactive online gaming [1, 2], are rapidly emerging. In general, these mobile applications demand considerable resources (e. g., high computing capacity and battery power) for real-time processing [1, 3, 4]. Unfortunately, due to the limited physical size of smart mobile devices, they are usually resource-limited to not efficiently support these applications, which incurs the conflict between the resource-hungry applications and resource-limited mobile devices.

To relieve the above conflict, mobile edge computing (MEC), a novel computing paradigm whose objective is to bring the computing and storage capacities close to mobile devices and users, becomes attractive [3] [5]. Therefore, smart mobile devices can offload some of their mobile application workloads via a wireless channel to nearby edge servers, which significantly augment the capacities of mobile devices [6]. Despite its advantage, when a large number of mobile users simultaneously offloads their computation tasks to an edge server, inefficiency resource allocation will not make full use of the limited resource and cause waste of

resource, resulting in low system performance (the weighted sum of the number of processed tasks, the number of punished tasks, and the number of dropped tasks). Therefore, how to allocate computing and communication resources efficiently to achieve performance optimization for the whole system is a very critical issue.

Most existing studies mainly focus on efficient resource allocation for achieving different optimization objectives (e.g., the total energy consumption and the processing delay) in MEC [7–12]. Particularly, in [7], an energy-efficient resource allocation scheme is designed to minimize the weighted sum energy consumption by optimizing the task operation sequences in a multiuser MEC system. In [8], an iterative resource allocation algorithm is proposed to minimize the total mobile devices' energy consumption and the total all tasks' completion time. In [9], the latency minimization scheme is proposed to minimize all tasks' computing and transmission time in a heterogenous MEC system. In [10], an ADMM-based algorithm is proposed to maximize the total network revenue while satisfying the QoS constraints. In [11], a sequential solution framework is proposed to tradeoff the power and latency by optimizing the active base station set, as well as computation resource allocation. In [12], an improved branch and bound method is designed to minimize the total latency of all tasks. Few works have paid attention to the impact of resource allocation of the edge server on system performance.

In this paper, we investigate the multiuser resource allocation problem in a MEC system, the goal of which is to maximize the long-term performance of the whole system. First, we establish a novel multiuser resource allocation system architecture which consists of an edge server and multiple mobile devices. Then, we construct the computation task model, task queueing model, and network model, respectively. Next, we formulate the multiuser resource allocation problem as a Markov Decision Process (MDP) whose state space mainly includes the number of arrival tasks, the workloads in the fond-end queue, the result data size in the back-end queue, and the transmission rate from the edge server to multiple mobile devices. According to the system state, an optimal resource allocation policy needs to be derived, and its objective is to maximize the long-term weighted sum of the number of processed tasks, the number of punished tasks, and the number of dropped tasks. Aiming to this problem, a deep deterministic policy gradient- (DDPG-) based performance-aware resource allocation (PARA) scheme is proposed. Finally, to evaluate the proposed PARA scheme, extensive experiments are conducted. The experimental results demonstrate that the PARA scheme can adaptively allocate the limited computational and communication resources to multiple mobile users and thus maximize the long-term weighted sum of the number of processed tasks, the number of punished tasks, and the number of dropped tasks. In addition, the performance of the PARA scheme outperforms the other benchmark policies, with respect to different parameter settings (such as the computing resource, the communication resource, the task workloads, and the result data size). We conclude the main contributions of this work as follows:

(i) We build a multiuser resource allocation architecture for computing workloads and transmitting result data to mobile devices, in which a data buffer consisting of two queues, namely, front-end queue and back-end queue, are used for each mobile user. The front-end queue is mainly applied for receiving the offloaded tasks; the back-end queue is mainly used for buffering the result data

(ii) We formulate the multiuser resource allocation problem as a MDP, the goal of which is to maximize the long-term performance of the whole system. Specifically, the system performance is the weighted sum of the number of processed tasks, the number of timeout tasks, and the number of dropped tasks

(iii) We propose a PARA scheme based on the DDPG algorithm to solve this formulation. The extensive experimental results demonstrate that the PARA scheme can achieve good effectiveness in the long-term performance of the whole system and outperforms the other three comparison algorithms

The remainder of the paper is organized as follows. In Section 2, the related works are reviewed. In Section 3, the system model and problem formulation are described. In Section 4, a multiuser resource allocation problem is formulated. In Section 5, the PARA scheme-based DDPG algorithm is presented. In Section 6, the experimental setup is elaborated and the experimental results are analyzed. Finally, in Section 7, this paper is concluded.

2. Related Works

It is very important to efficiently allocate resources to multiple mobile users in the MEC system. In recent years, a lot of studies have focused on this problem. The main goals of resource allocation are usually to minimize energy consumption or latency or both two of them. In terms of the optimization goals, these studies can be divided into three categories: (i) energy-efficient resource allocation, (ii) latency-based resource allocation, and (iii) energy consumption and latency-based resource allocation.

For energy-efficient resource allocation, [3, 7, 10–12] have studied some strategies to minimize the energy consumption of mobile devices or the whole system. Specially, [3] designs a low-complexity algorithm to solve the optimal task offloading sequence. In [7], an energy-efficient resource allocation scheme is designed to minimize energy consumption by optimizing the task operation sequences. [10] derives a threshold-based offloading strategy to minimize the weighted sum energy consumption. [11] designs an online task offloading algorithm to minimize the mobile devices' energy consumption under the execution delay constraint. In [12], the novel time allocation strategies are derived to optimize the resource allocation in a multiuser wireless-powered communication system, the objective of which is to maximize energy efficiency and throughout. In [13], a nonorthogonal multiple access-based offloading scheme is proposed to minimize the total energy consumption. In

Table 1: Major notations.

Symbols	Semantics
n	The number of mobile devices in MEC
T_{slot}	The time slot duration
W	The task workload
D	The result data size per unit workload
λ_i	The arrival rate of mobile device MD_i
Q_i	The data buffer for mobile device MD_i
L_i	The size of data buffer Q_i
$a_i(\tau)$	The number of arrival tasks for MD_i in time slot τ
$b_i(\tau)$	The number of computed tasks for MD_i in time slot τ
$h_i(\tau)$	The number of transmitted computation results for MD_i
$Q_i^F(\tau)$	The front-end queue of MD_i
$Q_i^B(\tau)$	The back-end queue of MD_i
$r_i(\tau)$	The transmission rate of a wireless channel from the edge server to MD_i
C	The process capacity with a single CPU core
N_{core}	The number of CPU cores
$p_i(\tau)$	The number of timeout tasks in time slot τ
$d_i(\tau)$	The number of dropped tasks in time slot τ
$R(\tau)$	The immediate reward in time slot τ

[14], a low-complexity algorithm is proposed to jointly optimize the user association, power control, computation capacity allocation, and location planning, the goal of which is to minimize the total power of both UEs and UAVs under latency and converge constraints.

For latency-based resource allocation, the objective is to minimize the execution delay. In particular, in [15], a software-defined task offloading strategy is proposed to minimize the delay by jointly optimizing the task placement and resource allocation. [16] investigates the multiuser resource allocation problem and designs the closed-form expressions to minimize the latency. [17] formulates the task offloading problem as a stochastic optimization problem in D2D communications. To solve this problem, a Steerable Economic ExPense Algorithm (SEEP) is proposed to minimize the expense and latency. In [18], a nonorthogonal multiple access scheme is proposed to solve the problem of multiuser task offloading. [19] designs the optimal solution for the problem of task offloading volume and resource allocation.

For energy consumption and latency-based resource allocation, the objective is to optimize the weighted sum of the energy consumption and latency. For example, [5] formulates the task offloading problem as a MDP and adopts a deep reinforcement learning technique to solve this problem. [20] jointly optimizes the offloading decision and computational resource allocation. In [21], a computation offloading and resource allocation scheme is designed to minimize the weighted sum of delay and energy consumption. [22] formulates the problem of task offloading and resource allocation as mixed-integer programming and then designs an iterative search algorithm to solve it. [23] decomposes the problem of task offloading and uplinks power allocation as two subproblems and then adopts a heuristic algorithm combined with convex optimization to solve it. In [24], the problem of task offloading and resource allocation is formulated as a RL-based theme, and a reinforcement learning-based optimization framework is proposed to minimize energy consumptions and latency. [25] formulates the task offloading problem as a MDP and then designs a decentralized task offloading strategy based on deep deterministic policy (DDPG) to solve it.

However, few works have paid attention to effectively allocating computing and communication resources to multiple mobile users in the case of insufficient resources. In addition, some works fail to take account of the resource allocation for transmitting the result data from an edge server to mobile users. Therefore, these strategies may not be suitable for applications with large-scale result data, such as AR, VR, and multiple media.

To solve this problem, in this paper, we mainly focus on performance-aware resource allocation, where the joint computation and communication resource can be effectively allocated in a multiuser time-varying MEC environment. We try to maximize the long-term weighted sum of the number of processed tasks, the number of punished tasks, and the number of dropped tasks.

3. System Model

In this section, the system architecture is first presented. Then, the computation task model, task queueing model, and network model are presented. For ease of understanding, the main notations used throughout this paper are given in Table 1.

3.1. System Architecture. As illustrated in Figure 1, a multiuser MEC system is comprised of an edge server and a set of n mobile devices $\text{MD} = \{\text{MD}_1, \cdots, \text{MD}_i, \cdots, \text{MD}_n\}$. The edge server can be denoted by a two-tuple $eNB = (C, N_{\text{core}})$, where C denotes the processor capacity with a single CPU core and N_{core} denotes the number of CPU cores. The edge server maintains a data buffer Q_i with size L_i for each MD_i. The data buffer Q_i consist of a front-end queue $Q_i^F(\tau)$ and a back-end queue $Q_i^B(\tau)$. The front-end queue $Q_i^F(\tau)$ is mainly used for receiving the offloaded tasks. The back-end queue $Q_i^B(\tau)$ is applied for buffering the result data. A discrete-time model is adopted for the MEC system, where the time horizon is logically divided to equal time slots and the time slot duration is T_{slot}. The index set of the time slots can be denoted by $\tau \in \mathcal{T} = \{0, 1, \cdots\}$. At the beginning of each time slot, the edge server makes an optimal resource allocation decision. We present the model components and our problem as follows.

3.2. Computation Task Model. The computation task model widely used in the existing literature [10, 26] is adopted in this paper. According to it, a computation task can be characterized by a four-tuple $t = (T_{st}(\tau), W, D, T_{\max})$, in which $T_{st}(\tau)$ denotes the task t arrives at the edge server at the τth time

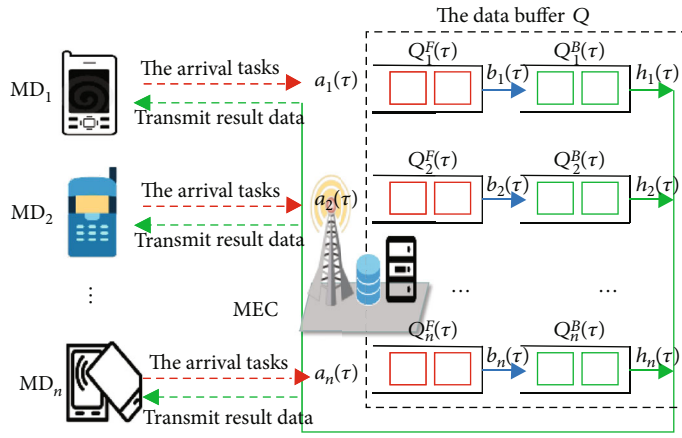


Figure 1: The multiuser resource allocation architecture.

slot, W denotes the task workload (GHz•s), D denotes the result data size (in MB) per unit workload, and $T_{\max}$ denotes the maximum tolerant latency which are required for task processing, respectively. In addition, we assume that the computation tasks' arrival process for the mobile device MD follows a Poisson distribution with a parameter $\lambda = (\lambda_1, \cdots, \lambda_i, \cdots, \lambda_n)$.

3.3. Task Queueing Models. At the tth time slot, each MD_i generate a series of independent and identically distributed (i.i.d) computation tasks. First, all or part of these computation tasks are offloaded to the edge server *eNB* and are buffered in front-end queue $Q_i^F(\tau)$. They can be executed from the $(\tau+1)$th time slot. The number of tasks arriving at $Q_i^F(\tau)$ can be denoted by $a_i(\tau)$. Let $\mathbf{A}(\tau) = (a_1(\tau), \cdots, a_i(\tau), \cdots, a_n(\tau))$ denote the task arrival vector. Since the size of the data buffer Q_i does not exceed L_i, the arrival tasks will be partially dropped when the data buffer has no sufficiently available space to accommodate them. Let $d_i(\tau)$ denote the number of dropped tasks. Then, the edge server allocates computing resource to execute tasks in the front-end queue $Q_i^F(\tau)$ and buffers the computation results in back-end queue $Q_i^B(\tau)$. The number of processed tasks by each MD_i can be denoted by $b_i(\tau)$. Let $\mathbf{B}(\tau) = (b_1(\tau), \cdots, b_i(\tau), \cdots, b_n(\tau))$ denote the vector of the number of processed tasks. At last, the edge server allocates communication resources to transmit the computation results to mobile devices. The number of computation results transmitted from *eNB* to MD_i can be denoted by $h_i(\tau)$. Let $\mathbf{h}(\tau) = (h_1(\tau), \cdots, h_i(\tau), \cdots, h_n(\tau))$ denote the vector of the number of transmitted computation results. Thus, according to the above descriptions, the front-end queue $Q_i^F(\tau)$ and back-end queues $Q_i^B(\tau)$ evolve according to the following equation:

$$\begin{aligned} Q_i^F(\tau+1) &= \max\left[Q_i^F(\tau) - b_i(\tau)W, 0\right] + \left[a_i(\tau) - d_i(\tau)\right]W, \\ Q_i^B(\tau+1) &= \max\left[Q_i^B(\tau) - h_i(\tau)D, 0\right] + b_i(\tau)D. \end{aligned} \tag{1}$$

3.4. Network Model. The computation results transmitting from an edge server to mobile users are performed by a wireless channel. Due to the impact of user mobility, the wireless channel gain state between them is dynamically varying across different time slots. Therefore, the efficiency of computation results transmitting highly depends on the wireless channel gain state. Let $r(\tau) = (r_1(t), \cdots, r_i(t), \cdots, r_n(t))$ denote the transmission rate of the wireless channel (i.e., data transmission capacity) from the edge server to n mobile users in time slot t. Specially, we can calculate $r_i(t)$ by Equation (2) [27].

$$r_i(t) = B(\tau)\log_2(1+\ ), \forall i \in \mathcal{N}, \tag{2}$$

where $B(\tau)$ is spectrum bandwidth of the edge server, P_{eNB} denotes the edge server's transmission power, σ^2 denotes the Gaussian white noise power, and G_i denotes the channel gain of MD_i. Besides, $G_i = \alpha(d_0/d_i)^\theta$, where α denotes the path-loss parameter, θ denotes the path-loss exponent, d_0 denotes the reference distance, and d_i denotes the distance between MD_i and edge server.

4. Performance-Aware Resource Allocation Problem Formulation

In this section, we formulate the multiuser resource allocation problem as an infinite discounted continuous state MDP. We first define the state and action spaces. Then, the reward function and constraints are defined. At last, the performance-aware resource allocation problem is formulated in detail.

4.1. State Space. At the beginning of each time slot, the system state $s(\tau) \in \mathbb{S}$ can be observed. It mainly consists of the number of arrival tasks, the workload in the front-end queue and the result data size in the back-end queue, and the transmission rate from the edge server to multiple mobile devices. It can be denoted by

$$s(\tau) = \Big[N_1^{arl}(\tau), \cdots, N_i^{arl}(\tau), \cdots, N_n^{arl}(\tau), Q_1^F(\tau), \cdots, Q_i^F(\tau), \cdots, Q_n^F(\tau), Q_1^B(\tau), \cdots, Q_i^B(\tau), \cdots, Q_n^B(\tau), r_1(\tau), \cdots, r_i(\tau), \cdots, r_n(\tau)\Big], \tag{3}$$

where $N_i^{arl}(\tau)$ denotes the number of tasks from a mobile device MD_i at the τ time slot, $r_i(\tau)$ denotes the transmission rate between the edge server and the ith mobile device at the τ time slot, $Q_i^F(\tau)$ denotes the remaining workloads in the ith front-end queue, and $Q_i^B(\tau)$ denotes the remaining result data size in the ith back-end queue at the τ time slot, respectively. According to the transfer equations of the front-end queue $Q^F(\tau)$ and back-end queues $Q^B(\tau)$, $Q_i^F(\tau)$ and $Q_i^B(\tau)$ can be dynamically evolved. At last, $r_i(\tau)$ can be dynamically calculated according to the network model.

4.2. Action Space. As we employ the resource share scheme of time-division multiplexing in this paper, an allocation action $a(\tau) \in \mathbb{A}$ consists of the time ratio at which different mobile users use computing and communication resources at each time slot τ. It can be denoted by

$$a(\tau) = [\mu_1(\tau), \cdots, \mu_i(\tau), \cdots, \mu_n(\tau), \eta_1(\tau), \cdots, \eta_i(\tau), \cdots, \eta_n(\tau)], \tag{4}$$

where $\mu_i(\tau)$ denotes the time ratio at which the ith mobile user occupies computing resource at the τth time slot, $\eta_i(\tau)$ denotes the time ratio at which the ith mobile user occupies a communication resource at the τth time slot. Particularly, the resource allocation can be elaborately optimized in a continuous action space, i.e., $\mu_i(\tau) \in [0, 1]$ and $\eta_i(\tau) \in [0, 1]$. Based on the current system state $s(\tau)$, the action $a(\tau) \in \mathbb{A}$ can be selected for each time slot $\tau \in \mathcal{T}$.

4.3. Reward Function. Since the resource allocation action is driven by the reward, the reward function plays an important role in the deep reinforcement learning algorithms. In this paper, the immediate reward $R(\tau)$ is the weighted sum of the number of processed tasks $h(\tau)$ in time slot τ, the number of punished tasks $p(\tau)$ in time slot τ, and the number of dropped tasks $d(\tau)$ in time slot τ. The tasks will be punished due to processing timeout, and the tasks will be dropped due to insufficient space of data buffer. The immediate reward $R(\tau)$ obtained by taking the action $a(\tau) \in \mathbb{S}$ at current system state $s(\tau) \in \mathbb{A}$ can be calculated by

$$R(\tau) = \omega_1 h(\tau) - \omega_2 p(\tau) - \omega_3 d(\tau), \tag{5}$$

where $h(\tau) = \sum_{i=1}^{n} h_i(\tau)$, $p(\tau) = \sum_{i=1}^{n} p_i(\tau)$, $d(\tau) = \sum_{i=1}^{n} d_i(\tau)$, ω_1, ω_2, and ω_2 are the weighted factors of the number of processed tasks $h_i(\tau)$, the number of punished tasks $p_i(\tau)$, and the number of dropped tasks $d_i(\tau)$, respectively. We further derive the $h_i(\tau)$, $p_i(\tau)$, and $d_i(\tau)$ as follows.

If the time ratio at which the computing resource is allocated to MD_i is $\mu_i(\tau) \in [0, 1]$ at tth time slot, the amount of executed computation tasks $b_i(\tau)$ can be denoted by Equation (6). The $\mu_i(\tau)$ must meet the constraints (7).

$$b_i(\tau) = \min\left\{\frac{\mu_i(\tau) T_{\mathrm{slot}} C}{W}, \frac{Q_i^F(\tau)}{W}\right\}, \tag{6}$$

$$\sum_{i \in N} \mu_i(\tau) \le 1. \tag{7}$$

If the time ratio at which the bandwidth resource is allocated to MD_i is $\eta_i(\tau) \in [0, 1]$, the amount of computation results transmitted $h_i(\tau)$ can be denoted by Equation (8). The $\eta_i(\tau)$ must meet the constraints (9).

$$h_i(\tau) = \min\left\{\frac{\eta_i(\tau) T_{\mathrm{slot}} r_i(\tau)}{D}, \frac{Q_i^B(\tau)}{D}\right\}, \tag{8}$$

$$\sum_{i \in N} \eta_i(\tau) \le 1. \tag{9}$$

Based on the above the description, due to that the maximum processing time of each task is set to be $T_{\max}$ time slots, the number of punished tasks $p_i(\tau)$ at τth time slot can be calculated by Equation (10). Moreover, due to that the data duffer's size is L_i, the number of dropped tasks $d_i(\tau)$ at τth time slot can be calculated by Equation (11).

$$p_i(\tau) = \max\left(\left(\sum_{\tau=1}^{\tau - T_{\max}} N_i^{\mathrm{arl}}(\tau) - \sum_{\tau=1}^{\tau} \frac{h_i(\tau)}{D}\right), 0\right), \quad \tau > 2, \tag{10}$$

$$d_i(\tau) = \max\left(N_i^{\mathrm{arl}}(\tau) - \left(L_i - Q_i^F(\tau) - Q_i^B(\tau)\right), 0\right). \tag{11}$$

We can see from Equation (10) that the better the channel condition, the larger the amount of data transferred and the lesser the remaining tasks in queues. To maximize the long-term reward, the optimal resource allocation policy is needed to derive.

4.4. Problem Formulation. In this paper, we aim to maximize the long-term reward by adaptively allocating the computing and communication resources to multiple mobile users. We formulate the multiuser resource allocation problem as follows.

$$\text{Maximum} : R(\tau) \tag{12}$$

$$\text{Subject to} : \sum_{i \in \mathcal{N}} \mu_i(\tau) T_{\mathrm{slot}} C \le C, \tag{13}$$

$$\sum_{i \in \mathcal{N}} \eta_i(\tau) T_{\mathrm{slot}} B \le B, \tag{14}$$

$$\sum_{i \in N} \mu_i(\tau) \le 1, \tag{15}$$

$$\sum_{i \in N} \eta_i(\tau) \le 1, \tag{16}$$

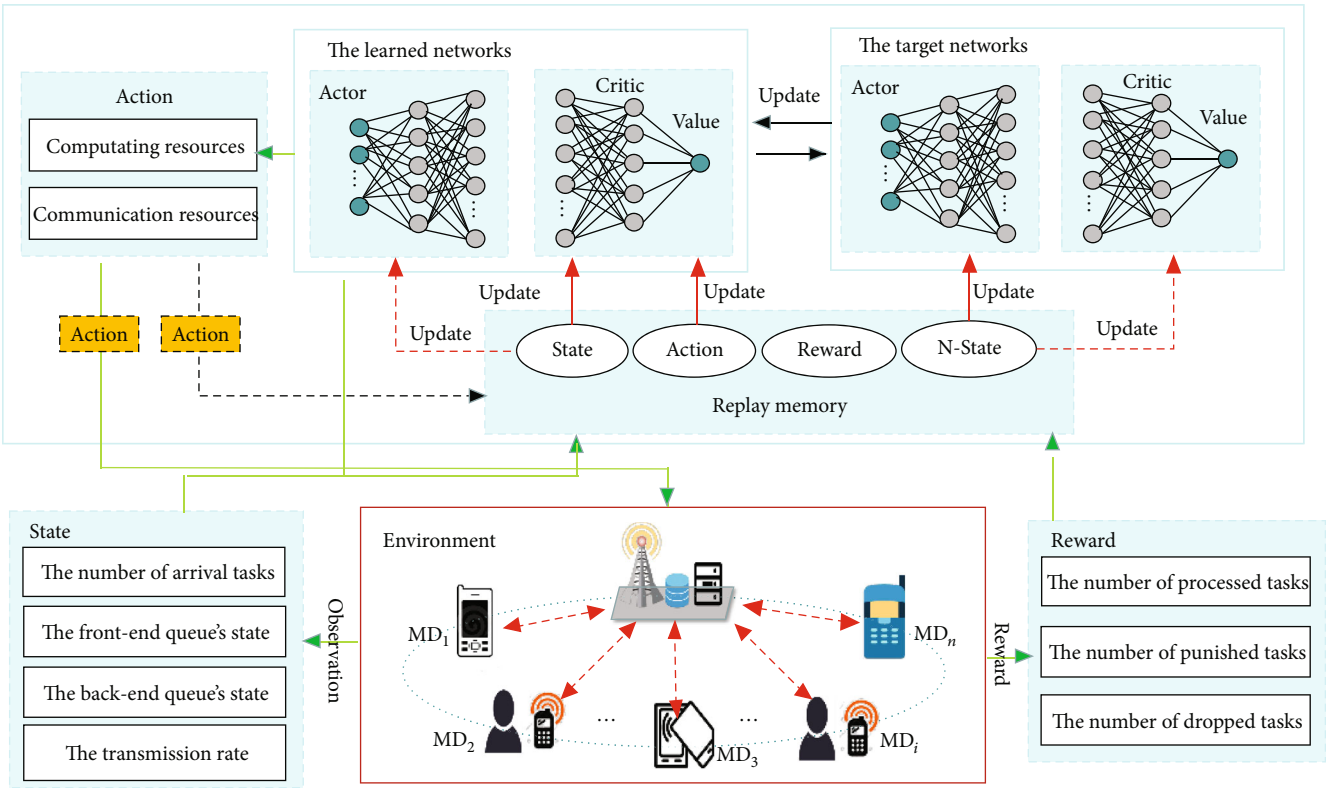


Figure 2: The PARA scheme based on the DDP algorithm.

where Equation (12) is the objective function of this paper. Equations (13) and (14) guarantee that the computing and communication resources that are allocated to n mobile users cannot exceed to the resource capacity of the edge server. Equation (15) denotes the time ratio at which the computing resource is allocated to n mobile users. Equation (16) denotes the time ratio at which the communication resource is allocated to n mobile users.

Lacking the prior knowledge of the MEC system, such as the number of mobile devices n, and statistics of task arrivals and wireless channel states, a model-free learning strategy is needed to tackle this kind of stochastic optimization problem. Since resource allocation is an optimization problem on the continuous action space, a performance-aware resource allocation scheme based on deep deterministic policy gradient (DDPG) [28] is proposed to solve this problem. In the next section, we will introduce the algorithm implementation.

5. Algorithm Implementation

The formulated multiuser resource allocation problem in this paper is essentially an infinite discounted MDP. Since the action space of the multiuser resource allocation problem contains plenty of continuous variables, the critical challenge to solve this problem is to make a decision in a continuous action space. Aiming to this problem, a performance-aware resource allocation (PARA) scheme based on DDPG is proposed. The PARA scheme can be illustrated in Figure 2.

As shown in Figure 2, we can observe that the DDPG algorithm mainly consists of three function components: (1) The learned networks: they mainly include an actor network $\mu(s(\tau) \mid \theta^{\mu})$ and a critic network $\mathcal{Q}(s(\tau), a(\tau) \mid \theta^{\mathcal{Q}})$. The actor network is used to choose an action $a(\tau)$ according to the current system state $s(\tau)$. The critic network is mainly adopted to estimate the action taken $a(\tau)$. (2) The target networks: they are a copy of the actor and critic networks, $\mu'(s = s(\tau) \mid \theta^{\mu'})$ and $\mathcal{Q}'(s(\tau), a(\tau) \mid \theta^{\mathcal{Q}'})$, respectively. They are mainly used to calculate the target value for training the learned networks. (3) Replay memory: the replay memory Ω is used to store the transition experience $(s(\tau), a(\tau), r(\tau), s^{-}(\tau))$ which includes current state $s(\tau)$, action taken $a(\tau)$, reward $r(\tau)$, and next state $s^{-}(\tau)$. The transition experience from replay memory is randomly chosen to train the learned networks and the target networks in the direction of minimizing a sequence of the loss functions. The detailed processes of DDPG can be illustrated as follows.

The system state $s(\tau)$ is first to feed into the actor network in the τth time slot. The actor network generates a resource allocation action $a(\tau)$ for the current system state $s(\tau)$. Based on the action $a(\tau)$, the edge server allocates the computing and communication resources to multiple mobile users to execute the task workloads and transmit the computation results to mobile users. Then, the immediate long-term reward can be calculated by Equation (4). Next, the resulting system state $s^{-}(\tau + 1)$ at next time slot $(\tau + 1)$ can be observed. At last, the transition experience $(s(\tau), a(\tau), R(\tau), s^{-}(\tau + 1))$ is stored into the experience replay memory Ω.

BEGIN

Initialize the learned networks $\mathcal{Q}(s, a \mid \theta^{\mathcal{Q}})$ and $\mu(s \mid \theta^{\mu})$ with weights $\theta^{\mathcal{Q}}$ and θ^{μ};
Initialize target networks $\mathcal{Q}'(s, a \mid \theta^{\mathcal{Q}'})$ and $\mu'(s \mid \theta^{\mu'})$ with weights $\theta^{\mathcal{Q}'} \longleftarrow \theta^{\mathcal{Q}}$, $\theta^{\mu} \longleftarrow \theta^{\mu'}$;
Initialize the experience replay memory Ω with size U, the minibatch $\tilde{\Omega} \subset \Omega$ with size $\tilde{U}$;
Initialize the task arrival rate λ, the data buffer's size L, the font-end queue $Q^F(t)$, the back-end queue $Q^B(t)$, and the transmission rate $r(\tau)$ between the edge server and the mobile users.
for curr_episode = 1, MAX_EPISODES **do**
Initialize a noise object *ounoise* to exploration actor;
Reset simulation parameters for performance-aware resource allocation environment, and observe an initial system state $s(\tau) \in \mathbb{S}$;
Initial the long-term reward $ep\text{-}reward = 0$;
  for each time slot $\tau = 1$, MAX_EP_STEPS **do**
Select an action $a(\tau) = \mu(s(\tau) \mid \theta^{\mu}) + ounoise$ based on the actor network and exploration noise;
Based on the action $a(\tau)$, the edge server allocates resource to execute workloads and transmits the computation results, and the immediate reward $\mathrm{R}(\tau)$ can be calculated, and then the next state $s(\tau + 1)$ can be observed;
The experience transition $(s(\tau), a(\tau), r(\tau), s^{-}(\tau + 1))$ is stored into the replay memory;
Update the long-term reward $ep\text{-}reward += \mathrm{R}(\tau)$ and system state $s(\tau) = s^{-}(\tau + 1)$;
A minibatch $\tilde{U}$ of transition experiences is randomly sampled from replay memory Ω;
Compute the target action value $\mu'(s^{-}(\tau + 1) \mid \theta^{\mu'})$ and the target Q value $\mathcal{Q}'(s^{-}(\tau + 1), \mu'(s^{-}(\tau + 1)) \mid \theta^{\mathcal{Q}'})$ for the next state $s^{-}(\tau + 1)$;
Update the target Q value $y(\tau) = \mathrm{R}(\tau) + \gamma \mathcal{Q}'(s^{-}(\tau + 1), \mu'(s^{-}(\tau + 1)) \mid \theta^{\mathcal{Q}'})$ for the current state $s(\tau)$;
Update the critic network $\mathcal{Q}(s(\tau), a(\tau) \mid \theta^{\mathcal{Q}})$ by minimizing the loss L:

$$L = 1/\tilde{U} \sum_{\tau} \left(y(\tau) - Q(s(\tau), a(\tau) \mid \theta^{Q})\right)^2$$

Update the actor network $\mu(s \mid \theta^{\mu})$ by using the sampled policy gradient:

$$\nabla_{\theta^{\mu}} J \approx 1/\tilde{U} \sum_{\tau} \nabla_{a(\tau)} Q(s(\tau), a(\tau) \mid \theta^{Q})\big|_{s=s(\tau), a=\mu(\tau)} \nabla_{\theta^{\mu}} \mu(s(\tau) \mid \theta^{\mu})\big|_{s(\tau)}$$

Update the target network:

$$\theta^{Q'} \longleftarrow T\theta^{Q} + (1 - T)\theta^{Q'}$$
$$\theta^{\mu'} \longleftarrow T\theta^{\mu} + (1 - T)\theta^{\mu'}$$

  end for
end for
END

Algorithm 1: PARA scheme.

In this training stage, a minibatch of transition experiences $\tilde{\Omega}$ is randomly sampled from the replay memory Ω to compute the target action value $\mu'(s(\tau + 1) \mid \theta^{\mu'})$ and target $\mathcal{Q}$ value $\mathcal{Q}'(s(\tau + 1), \mu'(s(\tau + 1)) \mid \theta^{\mathcal{Q}'})$ for the next system state $s^{-}(\tau + 1)$, respectively. Then, the target $\mathcal{Q}$ value for the system state $s(\tau)$ is updated to be $y(\tau) = R(\tau) + \gamma \mathcal{Q}'(s(\tau + 1), \mu'(s(\tau + 1)) \mid \theta^{\mathcal{Q}'})$ in the target critic network and transmits it to the critic network in the learned networks. After receiving y_i, the critic network in the learned network is updated by minimizing the loss function $L = \sum_{\tau}(y(\tau) - \mathcal{Q}(s(\tau), a(\tau) \mid \theta^{\mathcal{Q}}))^2/N$. Based on the $\theta^{\mathcal{Q}}$, the actor policy in the learned network is updated using the sampled policy gradient $\nabla_{\theta^{\mu}} \mathcal{J}$. In this way, after the training of MAX_EPISODES episodes, when the long-term reward converges, the optimal resource allocation can be gradually learned.

As for the testing stage, the network parameters learned in the training stage will firstly be loaded. Then, the edge server will start with an empty data buffer and interact with a randomly initialized environment. When its observation of the environment is obtained as the current state, the action can be selected according to the output of the actor network. Based on the action taken at the current system state, the reward can be calculated.

The PARA scheme is described in detail in Algorithm 1.

6. Simulation Experiments

In this section, we perform the simulation experiments to demonstrate the effectiveness of our proposed PARA scheme. First, we introduce the experiment parameter setup. Then, we introduce the experiment design including evaluation metrics and baseline algorithms. At last, we compare our proposed PARA scheme with the other three peer algorithms under different parameters and analyze the experimental results.

6.1. Experiment Setup. We consider a multiuser MEC system comprising an edge server and n mobile devices. Each mobile device can offload computation tasks to the edge server. When n mobile devices share the limited computing and communication resource of an edge server simultaneously, we propose the PARA scheme to solve optimal resource allocation policy. We implement our proposed PARA scheme and three baseline algorithms on python 3.6 and analyze corresponding experimental results. We set the experimental parameters referring to the literatures [1, 3, 29]. Major

simulation parameter setting in the following experiments is listed in Table 2.

Initially, the number of mobile devices is $n = 3$. We assume that the tasks' arrival process of each mobile device follows a Poisson distribution with mean $\lambda = (7, \cdots, 7, \cdots, 7)$. Moreover, the workload of each task follows the uniform distribution in the range [0.7, 1.1]Gycles. The result data size per unit workload follows the uniform distribution in the range [0.8, 1.2]MB. The processing time for each task cannot exceed to 2 time slots, which is $T_{\max} = 2$. The time slot length is $T_{\text{slot}} = 1$ s.

Initially, the processor cores of the edge server are $N_{\text{core}} = 10$. The computational capacity is $C = 2.5$ GHz with a single CPU core. In our experiments, we will adjust the computational resources by changing the number of CPU cores. Actually, the CPU frequency is not directly equal to the computing capacity, and however, they are proportional to each other. So, we assume that the computational resource is the CPU frequency, which is usually justified in practice and also has been used by [30]. The size of data buffer Q_i for mobile user MD_i is $L_i = 40$.

For the communication model, the transmission power of the edge server is $P_{eNB} = 40$ W, and the maximum channel bandwidth for edge server is $B = 100$ MHz, the additive white Gaussian noise power is $\sigma^2 = -174$ dbm/Hz, the path-loss constant is $\alpha = 0.01$, the path-loss exponent is $\theta = 4$, and the reference distance is $d_0 = 1$ m [1, 31, 32]. The distance between the mobile device and edge server is a random number which has the maximum value $d_{\max} = 350$ m.

The weights for the number of processed tasks, the number of punished tasks, and the number of dropped tasks are $\omega_1 = 5$, $\omega_2 = 1$, and $\omega_2 = 1$.

For the actor network in learned networks, there are two hidden layers, and each hidden layer is consisting of 30 neurons. These two hidden layers' activation functions are the Rectified Linear Unit (ReLU). As the value of the action is in the range of [0,1], the output layer's activation function is Tanh function. For the critic network in learned networks, it also consists of two hidden layers, and the number of neurons for each hidden layer is 30. The capacity of replay memory is set to be 10000, and the size of minibatch is set to be 64, and the training interval as 10.

6.2. Simulation Design

6.2.1. Evaluation Metrics

(1) The total reward: the immediate reward is the weighted sum of the number of immediate processed tasks $h(t)$, the number of immediate punished tasks $p(t)$, and the number of immediate dropped tasks $d(t)$. Therefore, the total reward can be calculated by Equation (4). We can use this metric to evaluate the quality of the resource allocation policy. The higher the quality of the resource allocation policy, the more fully the system resources can be utilized, the more tasks that can be processed, and the lesser tasks that can be punished and dropped

Table 2: Parameter setting.

Sections	λ_i	N_{core}	W	D	B	n
Section 6.3.1	[6, 10]	10	1.0	1.0	100	3
Section 6.3.2	7	[7, 11]	1.0	1.0	100	3
Section 6.3.3	7	10	[0.7,1.1]	1.0	100	3
Section 6.3.4	7	10	1.0	[0.8,1.2]	100	3
Section 6.3.5	7	10	1.0	1.0	[10, 100]	[3, 30]

(2) The total number of processed tasks: the number of immediate processed tasks can be calculated by Equation (7). Hence, the total number of processed tasks can be calculated by accumulating the number of immediate processed tasks. We can use this metric to measure how many are the total of processed tasks in a long time

(3) The total number of punished tasks: the number of immediate punished tasks can be calculated by Equation (9). The total number of punished tasks is the sum of the number of immediate punished tasks. We can use the metric to measure that the total number of tasks is punished in the system over a long time

(4) The total number of dropped tasks: the number of immediate dropped tasks can be calculated by Equation (10). The total number of dropped tasks can be calculated by accumulating the number of immediate dropped tasks. This matric is mainly used to measure that the total number of tasks is dropped in the system over a long time

6.2.2. Baseline Algorithm

(1) PF: this abbreviation stands for the proportional fair, which means this algorithm allocates the computing and communication resources according to the current lengths of the front-end queue and back-end queue. In this case, the computing and communication resources are allocated as follows: $\mu_i(t) = Q_i^F(t) / \sum_{i \in N} Q_i^F(t)$ and $\eta_i(t) = Q_i^B(t) / \sum_{i \in N} Q_i^B(t)$

(2) AF: this means the average resource allocation. This algorithm equally allocates the computing and communication resources to n mobile users. Therefore, computing and communication resource can be allocated as follows: $\mu_i(t) = 1/n$ and $\eta_i(t) = 1/n$

(3) RF: this represents the random resource allocation. The algorithm randomly allocates computing and communication resources to n mobile users. At each time slot, it randomly generates the computing resource's allocation rate $\mu_i(t) \in [0, 1]$ and the communication resource's allocation rate $\eta_i(t) \in [0, 1]$ for each mobile user MD_i and normalize them. Therefore, the computing and communication resource can be allocated as follows: $\mu_i(t) = \mu_i(t) / \sum_{i \in N} \mu_i(t)$ and $\eta_i(t) = \eta_i(t) / \sum_{i \in N} \eta_i(t)$

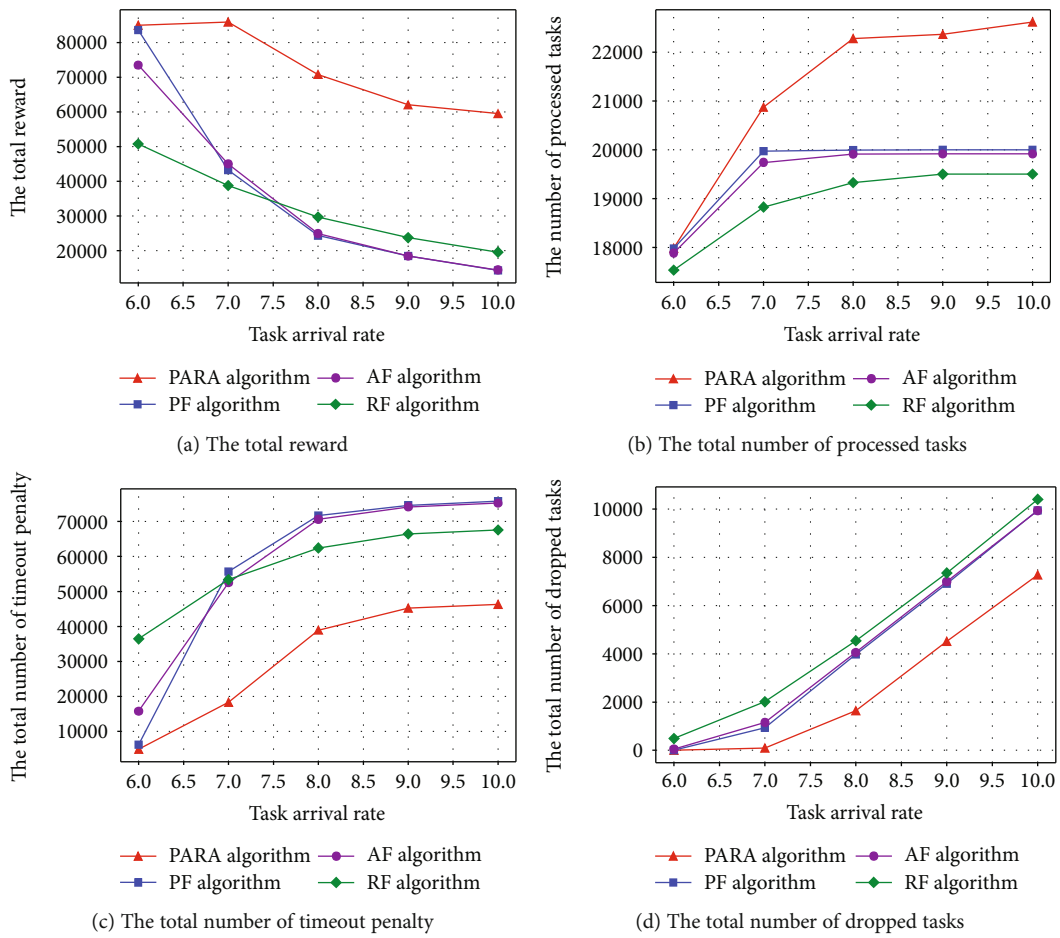


Figure 3: Performance impact of different arrival rates.

6.3. Performance Analysis

6.3.1. Performance Impact of Different Task Arrival Rates. To examine the performance impact of different task arrival rates, we vary the task arrival rate λ from 6 to 10 with an increment of 1. Figure 3(a) plots the long-term reward obtained by four algorithms under different task arrival rates. We can observe from Figure 3(a) that the PARA algorithm outperforms the other three algorithms. In particular, the long-term reward of the PARA algorithm increases first and then decreases, while the long-term rewards of the PF algorithm, AF algorithm, and RF algorithm decrease as the task arrival rate increase. That is because the PARA algorithm can adaptively allocate computing and communication resource according to the changing environment, such as the task arrival, the front-end queue's workload, the back-end queue's data size, and the transmission rate, which can make full use of the computing and communication resources to process much more tasks. However, the PF, AF, and RF algorithms fail to perceive the dynamic environment and reasonably allocate computing and communication resources. As the task arrival rate increases, the greater number of tasks arrives at the edge server, the greater number of tasks is processed by the PARA algorithm. However, the number of tasks that the PF, AF, and RF algorithms process increase first and then hardly changed. Figure 3(b) shows the total number of processed tasks in each episode. We can observe from Figure 3(b) that the PARA algorithm is higher than that of the PF, AF, and RF algorithm. In addition, Figures 3(c) and 3(d) show the total number of dropped tasks and the total number of punished tasks in each episode, respectively. We can see from these two figures that the total number of dropped tasks and the total number of punished tasks gradually increase with the increasing of the task arrival rate, and these obtained by the PARA algorithm are lower than the PF, AF, and RF algorithms. In this context, the long-term reward of the PF, AF, and RF algorithms gradually decreases, and that of the PARA algorithm increases first and then decreases.

6.3.2. Performance Impact of Different Computing Resource. The computing capacity of the edge server is mainly relative to the number of CPU cores. To investigate the performance impact of different computing capacity for the edge server, we vary the CPU cores from 7 to 11 with an increment of 1. Figure 4(a) plots the long-term rewards of all algorithms. We can observe from Figure 4(a) that the long-term rewards of all algorithms increase gradually with the increase of the CPU cores. This is because, with more computing resource, the greater number of tasks can be processed, and the fewer

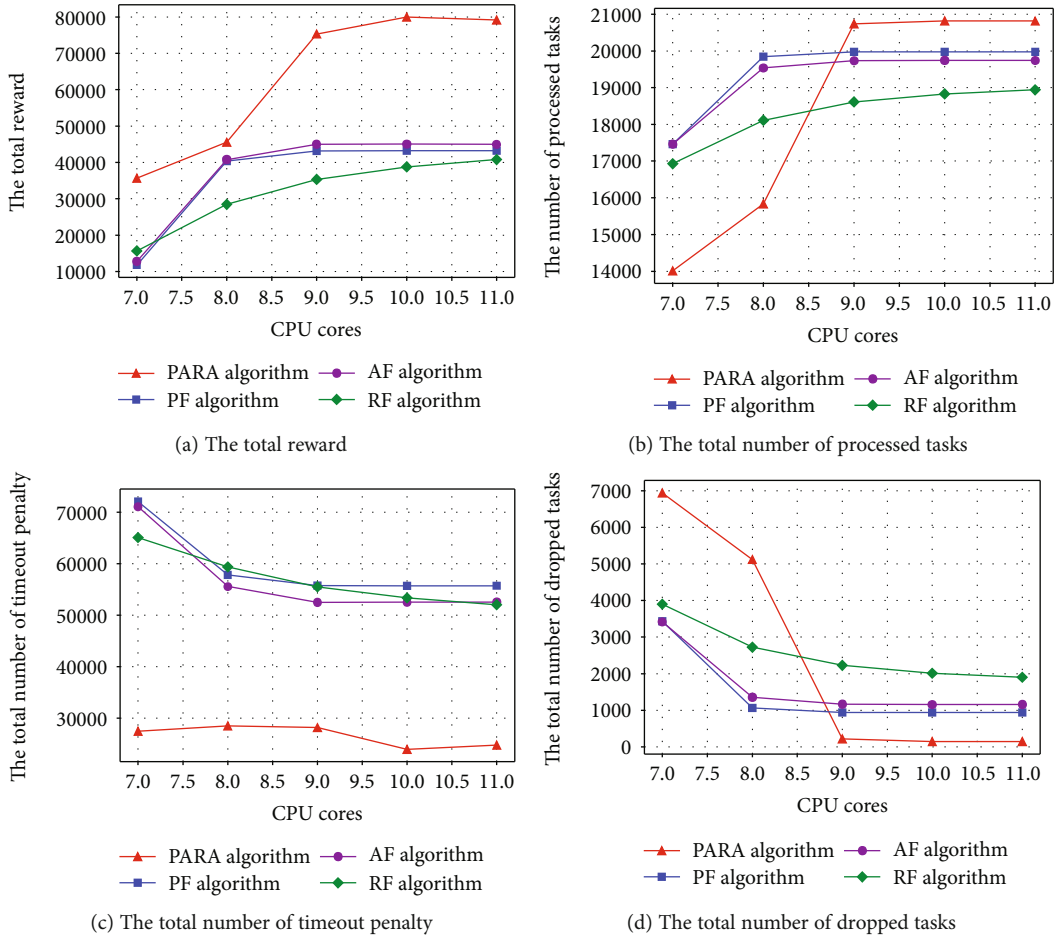


Figure 4: Performance impact of different computing resource.

number of tasks are punished and dropped, which leads to a higher long-term reward. It is noted that the curves of the PF, AF, and RF algorithms are flat when the number of CPU cores is equal to or larger than 8, while the curve of the PARA algorithm is flat when the number of CPU cores is equal to or larger than 10. This can be explained that the number of processed tasks reaches the maximum under this computing and communication capacity. Moreover, the long-term reward of the PARA algorithm is higher than that of the PF, AF, and RF algorithms. That is because the PARA algorithm can perceive the environment dynamics and adaptively allocate computing and communication resources to process much more tasks.

In Figure 4(b), below the 9 CPU cores, the fewer number of CPU cores, the lower computing capacity of the edge server, which leads to the fewer number of processed tasks, the greater number of timeout tasks and dropped tasks. To maximize the reward (the weighted sum of the number of processed tasks, the number of punished tasks, and the number of dropped tasks) of the PARA algorithm, it can decrease the penalty of task timeout and the number of dropped tasks. If it is a priority given to reduce the number of dropped tasks, it will incur much more penalty of tasks timeout due to the insufficient computing capacity, resulting in the lower reward. On the contrary, if it is a priority given to reduce the number of task timeout, it will drop more tasks, resulting in a higher reward for the PARA algorithm. The greater number of dropped tasks, the smaller number of processed tasks, and therefore, below the 9 CPU cores, the number of processed tasks for the PARA algorithm is lower than that for the PF, AF, and RF algorithms. Moreover, as shown in Figure 4(b), when the number of CPU cores is greater than or equal to 9, the computing capacity of the edge server is enough to process the arrival tasks while satisfying the execution delay constraint and thereby the number of timeout tasks and dropped tasks decrease and the number of processed tasks increase.

As shown in Figures 4(c) and 4(d), the total number of punished tasks and the total number of dropped tasks in each episode gradually decrease with the increasing number of CPU cores. Moreover, the total number of punished tasks and the total number of dropped tasks for the PARA algorithm are lower than that of the PF, AF, and RF algorithms. The reason is that the PARA algorithm can make full use of the computing and communication resources to process much more tasks.

6.3.3. Performance Impact of Different Workloads. To investigate the impact of different task workloads on the long-term reward, we conduct the experiments with this parameter

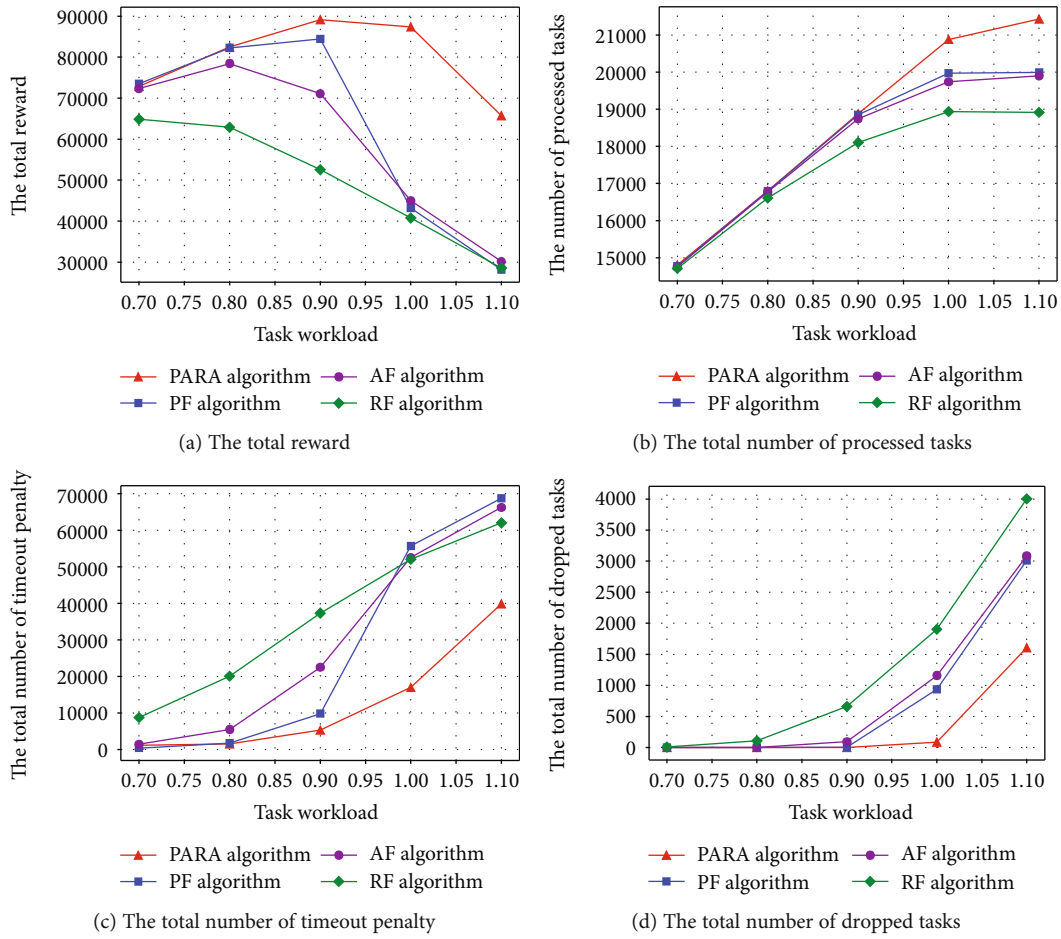


Figure 5: Performance impact of different workloads.

varying from 0.7 to 1.1 GHz•s with an increment of 0.1. The related results are plotted in Figure 5. We can see from Figure 5(a) the long-term rewards of the PARA, PF, and AF algorithms first increase and then gradually decrease with the increasing of task workload, while the long-term reward of the RF algorithm gradually decreases. The reason is that with the increasing of task workload, the greater number of tasks is processed by all algorithms. However, when the workload is equal to or larger than 0.9, the greater number of tasks is dropped and punished, and thereby, the long-term rewards gradually decrease. Moreover, the long-term reward of the PARA algorithm is higher than that of the PF, AF, and PARA algorithms. That is because the PARA algorithm can process much more tasks than the PF, AF, and RF algorithms. At the same time, the PARA algorithm drops and punishes less tasks than the PF, AF, and RF algorithms, which leads to the higher long-term rewards of the PARA algorithm.

Figure 5(b) shows that with the increase of task workload, the total number of processed tasks in each episode is gradually increasing and tend to be stable. The reason is that the larger task workload, the lager the data size of generated computation result, the greater number of processed tasks. Moreover, the total number of processed tasks for the PARA algorithm is higher than that for the PF, AF, and RF algorithms. That is because the PARA algorithm can make full use of the resource to process much more tasks. Figures 5(c) and 5(d) show that with the increase of task workload, the total number of punished tasks and the total number of dropped tasks gradually increase. The reason is that with the limited resource of the edge server, the number of processed tasks gradually increase and tend to be stable with the increase of task workload, which leads to the number of punished and dropped tasks gradually increase. In addition, the total number of punished tasks and the total number of dropped tasks for the PARA algorithm are lower than that of the PF, AF, and RF algorithms. The reason is the same as above.

6.3.4. Performance Impact of Different Result Data Sizes. Figure 6 illustrates the impact of result data size on the long-term reward. We discuss about the performance of all algorithms when the result data size is varied from 0.8 to 1.2 with the increase of 0.1. As observed from Figure 6(a), the long-term rewards of all algorithms first gradually increase and then tend to be stable under different result data sizes. That is because with the limited computing and communication resources, the maximum number of processed tasks is constant. With the result data sizes gradually increase, the number of processed tasks first gradually

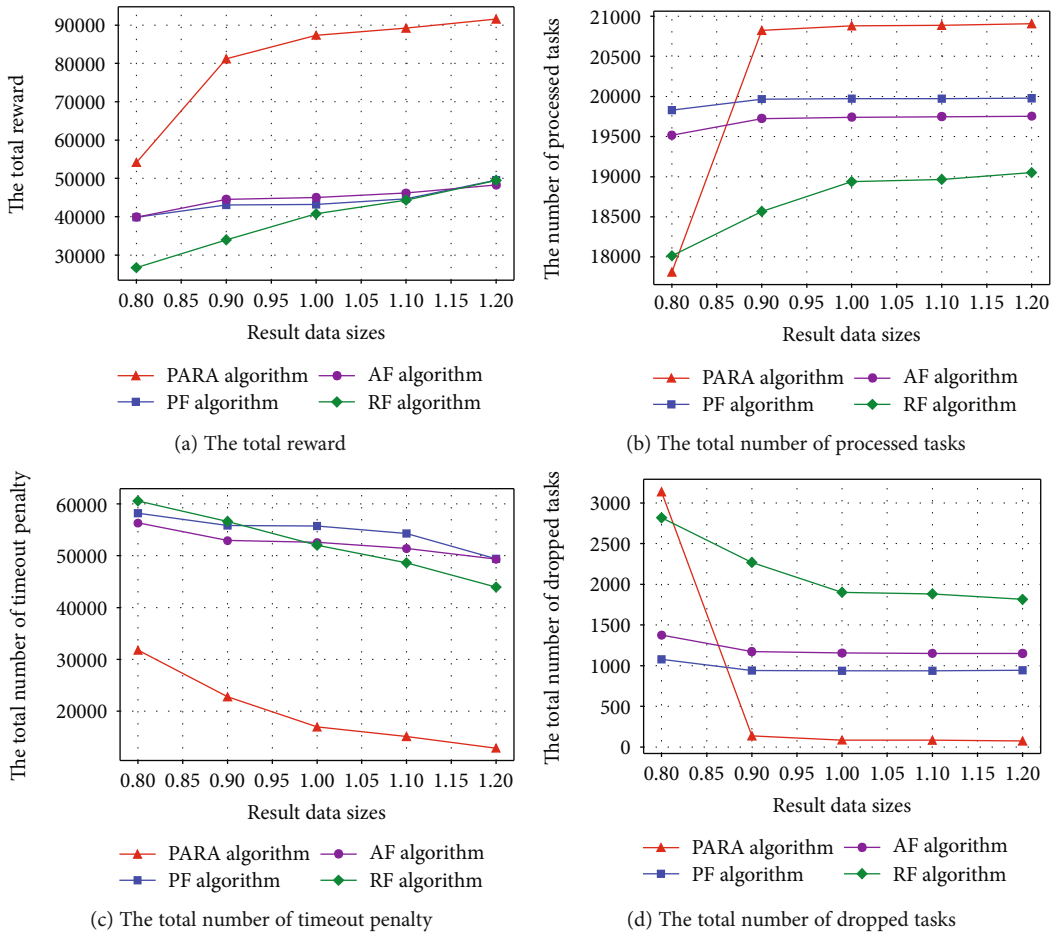


Figure 6: Performance impact of different result data sizes.

increase and then hardly change, and thereby, the number of dropped tasks and the number of punished tasks first gradually decrease and then hardly changed. In particular, the long-term reward of the PARA algorithm outperforms that of the PA, AF, and RF algorithms. The reason is that the PARA algorithm can perceive the environment dynamics and make full use of the resources to process much more tasks, and thereby, the long-term reward is higher than that of the PA, AF, and RF algorithms. As shown in Figure 6(b), we observe that the number of processed tasks first increases and then tend to be stable with the increase of the result data size. Moreover, the total number of processed tasks for the PARA algorithm is higher than that for the PF, AF, and RF algorithms. Figures 6(c) and 6(d) show the total number of punished tasks and the total number of dropped tasks, respectively. We can see from these two figures that the total number of dropped tasks and the total number of punished tasks gradually decrease and tend to stable with the increase of the result data size, and these obtained by the PARA algorithm are lower than the PF, AF, and RF algorithms. In this context, the long-term reward of the PARA algorithm is higher than that of the PF, AF, and RF algorithms.

6.3.5. Performance Impact of Different Bandwidths. To investigate the impact of bandwidth on the long-term reward, we conduct the experiments with this parameter varying from 20 to 100 with the increase of 20. The related results are shown in Figure 7. We can see from Figure 7(a) that as the bandwidth goes from 20 to 100, the long-term rewards of all algorithms increase gradually. This is because the larger bandwidth is, the higher transmission rate for each mobile device is, which makes the edge server process more tasks in each time slot, resulting in fewer number of punished and dropped tasks. Moreover, we can also observe from Figure 7(a) that the long-term reward of the PARA algorithm is higher than that of the PF, AF, and RF algorithms. The main reason is that the PARA algorithm can give an optimal computing and communication resource allocation scheme according to the environment dynamic, and hence, the PARA algorithm can make full use of computing and communication resource to process more tasks in contrast with the PF, AF, and RF algorithms. Due to that the PARA algorithm can process more tasks, it drops and punishes lesser tasks, which results in the higher long-term rewards of the PARA algorithm.

Figure 7(b) shows that the total number of processed tasks in each episode. We can observe from Figure 4(b) that the number of processed tasks for all algorithms gradually increases with the increase of bandwidth, and the number of processed tasks for the PARA algorithm is higher than that

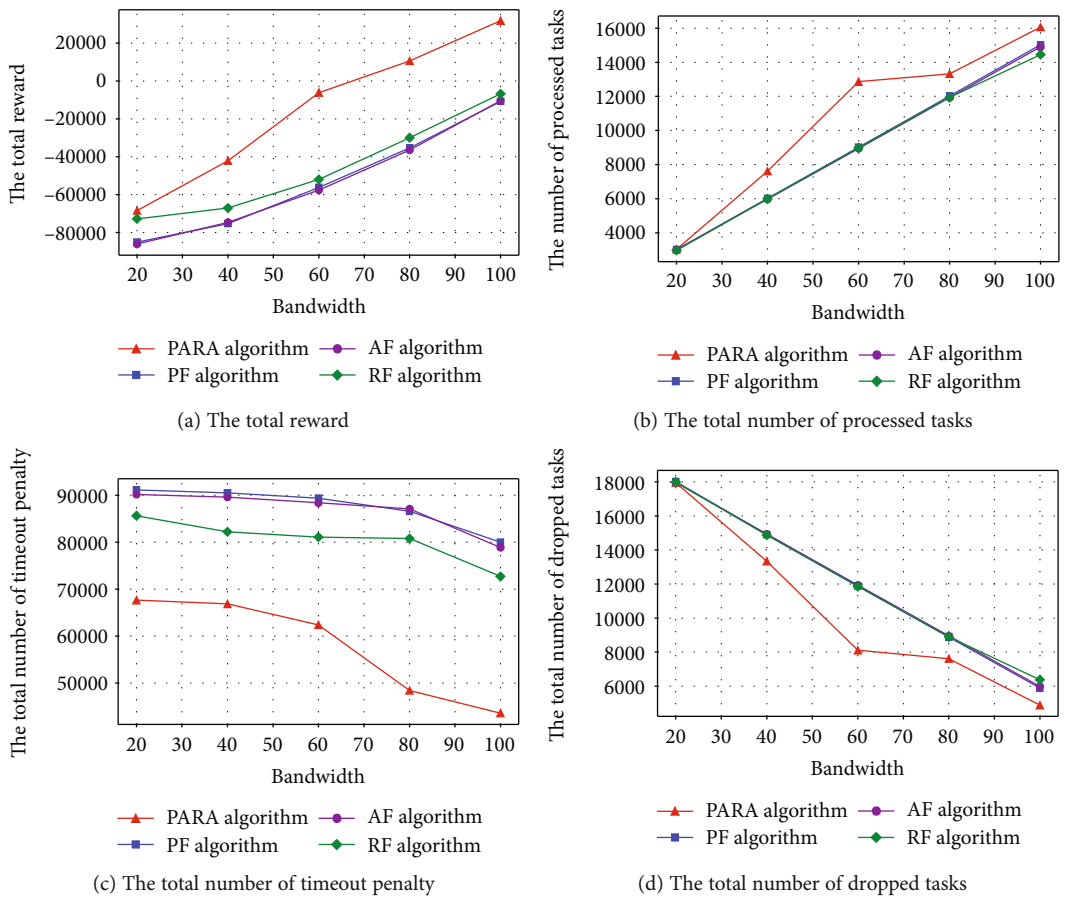
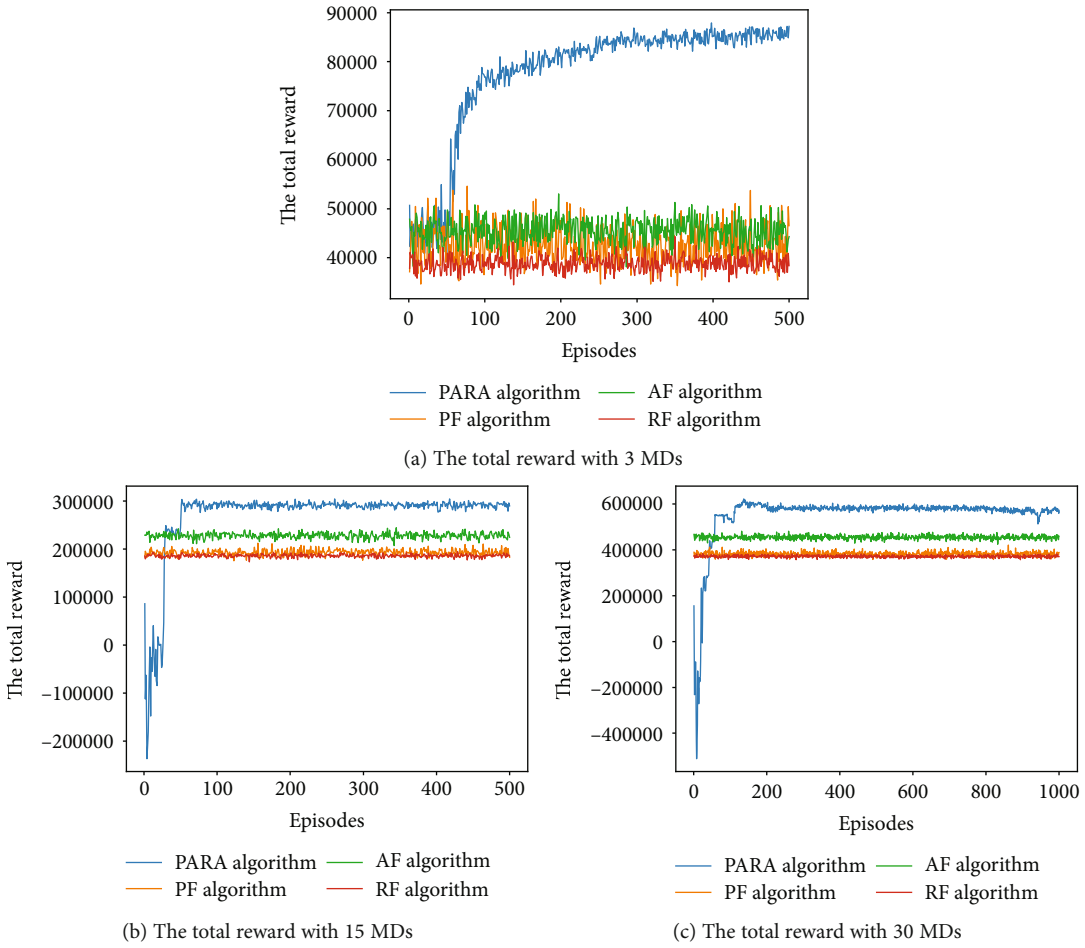


Figure 7: Performance impact of different bandwidths.

for the PF, AF, and RF algorithms. The reason is explained as above. As shown in Figures 4(c) and 4(d), we can further observe that with the increase of bandwidth, the number of punished tasks and the number of dropped tasks in each episode gradually decrease. That is because the larger the bandwidth is, the more tasks are processed, the fewer tasks are punished and dropped. In addition, the number of punished tasks and the number of dropped tasks for the PARA algorithm are lower than that of the PF, AF, and RF algorithms. The reason is that the PARA algorithm can make full use of the computing and communication resource to process much more tasks.

6.3.6. Performance Impact of Different Numbers of MDs. For the purpose of revealing the performance impact of the number of mobile users, we conduct the experiments with 3, 15, and 30 mobile users, respectively. The related results are given in Figure 8. We can observe from Figures 8(a)–8(c) that the performance of the PARA algorithm outperforms the PF, AF, and RF algorithms. This is because, under limited computing and communication resources, the PARA algorithm can perceive the environment dynamics and adaptively allocate computing and communication resources to process more tasks, to drop and punish less tasks. In addition, we can observe from Figure 8 that the long-term rewards of all algorithms increase with the number of mobile users and the resource of the edge server. The least long-term reward was obtained by 3 mobile users, a moderate level of long-term reward was obtained by 15 mobile users, and the most long-term reward was obtained by the 30 mobile users. The experiments show that the PARA algorithm can achieve excellent performance in different scale scenarios.

6.3.7. Analysis of PARA Scheme's Performance. Figures 9(a)–9(d) show the learning curves of the PARA scheme over variations of task arrival rate, computing resource, task workloads, and result data size, respectively. We can observe from Figure 9(a) that the long-term rewards of different task arrival rates gradually increase and then tend to be stable with the increasing of learning time (i.e., the number of episodes) from 1 to 500. The result indicates that the proposed PARA scheme can converge to an optimal policy to maximize the long-term reward. In addition, the long-term reward with the task arrival rate 6 is the largest, the task arrival rate 10 is the lowest, and the task arrival rate 8 is between 6 and 10. The reason is that with the increasing of task arrival rate, the number of arrival tasks increases. Due to the limited resource of the edge server, the greater number of arrival tasks, the greater number of processed tasks, the greater number of punished tasks, and the greater number of dropped tasks. However, the long-term weighted sum of the number of processed tasks, the number of punished tasks, and the number of punished tasks decreases.



(a) The total reward with 3 MDs

(b) The total reward with 15 MDs

(c) The total reward with 30 MDs

Figure 8: Performance impact of different number of MDs.

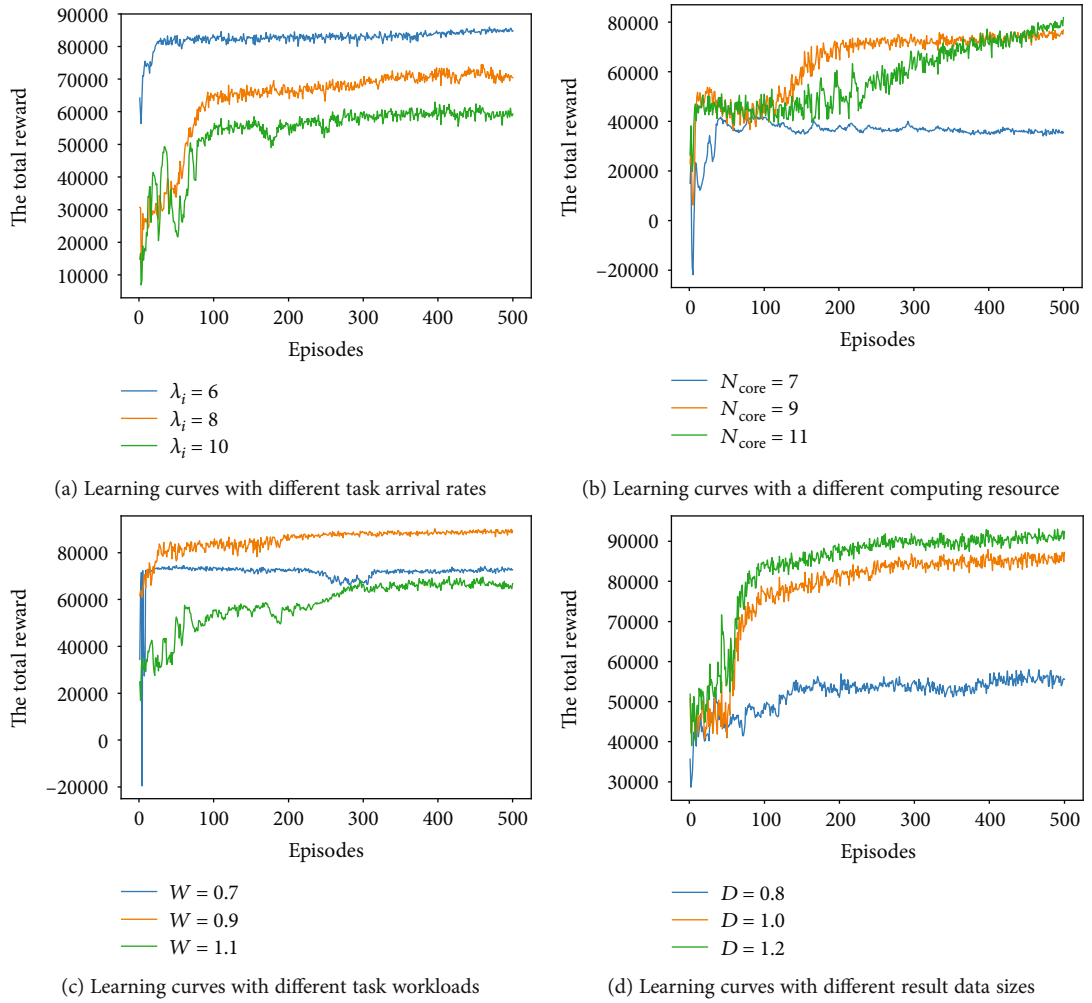
As shown in Figure 9(b), with the increasing of the number of learning episode, the long-term rewards of the different number of CPU cores gradually increase and become stable. It means that the PARA scheme can find the optimal allocation policy under the different number of CPU cores. Moreover, the long-term reward of the number of CPU cores 7 is the lowest, the number of CPU cores 11 is the largest, and the number of CPU cores 9 is between 7 and 11. That is because the greater number of CPU cores, the greater number of processed tasks, and the fewer number of punished tasks, and the fewer number of dropped tasks. Therefore, the long-term weighted sum increases.

Figure 9(c) shows that the long-term reward of different workloads first increases and then becomes stable. The results demonstrate that our proposed PARA scheme can converge to an optimal policy. In addition, we can observe from Figure 9(c) that long-term reward first increases and then reduces with the increasing of workload. Specifically, the long-term reward of workload 0.9 is the largest, the long-term reward of workload 1.1 is the lowest, and the long-term reward of workload 0.7 is between 0.9 and 1.1. It is because when the workload is less than 0.9, our proposed PARA scheme can process the greater number of tasks with the increasing of task workload. However, as the workload increases further, the number of dropped tasks and the number of punished tasks increase, and thereby, the long-term weighted sum decreases.

Figure 9(d) shows that the long-term rewards of different result data sizes gradually increase and tend to be stable. The result indicates that the PARA scheme can converge under different result sizes. Moreover, from Figure 9(d), we can see that the long-term reward of result data size 1.2 is the largest, the long-term reward of result data size 0.8 is the lowest, and the long reward of result data size 1.0 is between 0.8 and 1.2. It is due to that when the result data size increases, the number of processed tasks first increases. However, when the transferred result data size reaches the maximum that the system can process, the number of processed tasks hardly changes. Therefore, the long-term rewards first increase and then hardly changed.

7. Conclusions and Future Work

In this paper, we investigate the problem of joint computing and communication resource allocation in a multiuser MEC system. Aiming at this problem, we first build a computing and communication resource allocation architecture for multiple users. Then, we formulate this problem as an



(a) Learning curves with different task arrival rates

(b) Learning curves with a different computing resource

(c) Learning curves with different task workloads

(d) Learning curves with different result data sizes

FIGURE 9: Learning curves with different performance parameters.

infinite discounted continuous state MDP. At last, the front-end and back-end queues are used for each mobile user, and a PARA scheme based on DDPG is proposed to solve the optimal resource allocation policy, the objective of which is to maximize the long-term weighted sum of the number of processed tasks, the number of punished tasks, and the number of dropped tasks. To demonstrate the effectiveness of our proposed PARA scheme, we conduct extensive experiments and compare the PARA scheme with the other three algorithms, such as the random fair (RF) algorithm, average fair (AF) algorithm, and proportional fair (PF) algorithm. The experimental results demonstrate that the PARA scheme is more effective than the RF, AF, and PF algorithms under the different parameters, such as the task arrival rate, the computing capacity, the task workload, the data size of computation result, and the number of mobile users. Therefore, this work can provide valued guidelines for the practical multiuser resource allocation in the MEC system. In future work, we will further investigate the distributed learning approach such as federated learning which is exploited to decrease the complexity at one central node.

Data Availability

(1) The experiment data supporting this experiment analysis are from previously reported studies, which have been cited. (2) The experiment data used to support the findings of this study are included within the article. (3) The experiment data are described in Section 6 in detail.

Conflicts of Interest

The authors declare that there is no conflict of interest.

Acknowledgments

This work was supported by the National Science Foundation of China (Nos. 61802095, 61572162, and 61572251), the Zhejiang Provincial National Science Foundation of China (Nos. LQ19F020011 and LQ17F020003), the Zhejiang Provincial Key Science and Technology Project Foundation (No. 2018C01012), and the Open Foundation of State key Laboratory of Networking and Switching Technology

(Beijing University of Posts and Telecommunications) (No. SKLNST-2019-2-15).

References

[1] Y. Mao, J. Zhang, S. H. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, 2017.

[2] H. Tang, C. Li, J. Bai, J. H. Tang, and Y. Luo, "Dynamic resource allocation strategy for latency-critical and computation- intensive applications in cloud-edge environment," *Computer Communications*, vol. 134, pp. 70–82, 2019.

[3] X. Wang, Y. Cui, Z. Liu, J. Guo, and M. Yang, "Optimal resource allocation for multi-user {MEC} with arbitrary task arrival times and deadlines," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pp. 1–6, Shanghai, China, 2019.

[4] C. Yi, J. Cai, and Z. Su, "A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications," *IEEE Transactions on Mobile Computing*, vol. 19, no. 1, pp. 29–43, 2020.

[5] L. Lei, H. Xu, X. Xiong, K. Zheng, W. Xiang, and X. Wang, "Multi-user resource control with deep reinforcement learning in IoT edge computing," 2019, http://arxiv.org/abs/1906.07860.

[6] M. T. Beck, M. Werner, S. Feld, and S. Schimper, "Mobile edge computing: a taxonomy," in *Proceedings of the Sixth International Conference on Advances in Future Internet*, pp. 48–55, Lisbon, Portugal, 2014.

[7] J. Guo, Z. Song, Y. Cui, Z. Liu, and Y. Ji, "Energy-efficient resource allocation for multi-user mobile edge computing," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pp. 1–7, Singapore, 2017.

[8] J. Zhang, X. Hu, Z. Ning et al., "Joint resource allocation for latency-sensitive services over mobile edge computing networks with caching," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4283–4294, 2019.

[9] Z. Yang, C. Pan, J. Hou, and M. Shikh-Bahaei, "Efficient resource allocation for mobile-edge computing networks with NOMA: completion time and energy minimization," *IEEE Transactions on Communications*, vol. 67, no. 11, pp. 7771–7784, 2019.

[10] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp. 1397–1411, 2017.

[11] Y. Mao, J. Zhang, S. H. Song, and K. Ben Letaief, "Power-delay tradeoff in multi-user mobile-edge computing systems," in *2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, Washington, DC, USA, 2016.

[12] T. A. Zewde and M. C. Gursoy, "Optimal resource allocation for energy-harvesting communication networks under statistical QoS constraints," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 2, pp. 313–326, 2019.

[13] Y. Pan, M. Chen, Z. Yang, N. Huang, and M. Shikh-Bahaei, "Energy-efficient NOMA-based mobile edge computing offloading," *IEEE Communications Letters*, vol. 23, no. 2, pp. 310–313, 2019.

[14] Z. Yang, C. Pan, K. Wang, and M. Shikh-Bahaei, "Energy efficient resource allocation in UAV-enabled mobile edge computing networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 9, pp. 4576–4589, 2019.

[15] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[16] J. Ren, G. Yu, Y. Cai, and Y. He, "Latency optimization for resource allocation in mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, no. 8, pp. 5506–5519, 2018.

[17] J. Feng, L. Zhao, J. Du, X. Chu, and F. R. Yu, "Computation offloading and resource allocation in D2D-enabled mobile edge computing," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6, Kansas City, MO, 2018.

[18] Y. Wu, L. P. Qian, K. Ni, C. Zhang, and X. Shen, "Delay-minimization nonorthogonal multiple access enabled multi-user mobile edge computation offloading," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 3, pp. 392–407, 2019.

[19] T. Bai, C. Pan, Y. Deng, M. Elkashlan, and A. Nallanathan, "Latency minimization for intelligent reflecting surface aided mobile edge computing," 2019, http://arxiv.org/abs/1910.07990.

[20] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, pp. 1–9, Atlanta, GA, 2017.

[21] 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 Transactions on Communications*, vol. 66, no. 4, pp. 1594–1608, 2018.

[22] J. Zhang, X. Hu, Z. Ning et al., "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2633–2645, 2018.

[23] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2019.

[24] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *2018 IEEE Wireless Communications and Networking Conference (WCNC),*, pp. 1–6, Barcelona, Spain, 2018.

[25] Z. Chen and X. Wang, "Decentralized computation offloading for multi-user mobile edge computing: a deep reinforcement learning approach," 2018, http://arxiv.org/abs/1812.07394.

[26] E. Meskar, T. D. Todd, D. Zhao, and G. Karakostas, "Energy aware offloading for competing users on a shared communication channel," *IEEE Transactions on Mobile Computing*, vol. 16, no. 1, pp. 87–96, 2017.

[27] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 17, no. 3, pp. 1784–1797, 2017.

[28] T. P. Lillicrap, J. J. Hunt, A. Pritzel et al., "Continuous control with deep reinforcement learning," 2015, http://arxiv.org/abs/1509.02971.

[29] X. Qiu, L. Liu, W. Chen, Z. Hong, and Z. Zheng, "Online deep reinforcement learning for computation offloading in

blockchain-empowered mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 8050–8062, 2019.

[30] X. Lyu, W. Ni, H. Tian et al., "Optimal schedule of mobile edge computing for internet of things using partial information," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2606–2615, 2017.

[31] W. Jiang, G. Feng, S. Qin, and T. S. P. Yum, "Efficient D2D content caching using multi-agent reinforcement learning," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 511–516, Honolulu, Hawaii, 2018.

[32] B. Huang, Y. Li, Z. Li et al., "Security and cost-aware computation offloading via deep reinforcement learning in mobile edge computing," *Wireless Communications and Mobile Computing*, vol. 2019, Article ID 3816237, 20 pages, 2019.