Computer Networks 215 (2022) 109181

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Resource allocation for MEC system with multi-users resource competition based on deep reinforcement learning approach

Bin Qu [a,b], Yan Bai [c], Yul Chu [d], Li-e Wang [a,b], Feng Yu [a,b,e], Xianxian Li [a,b,*]

[a] Guangxi Key Lab of Multi-source Information Mining & Security, Guangxi Normal University, Guilin, 541004, China
[b] School of Computer Science and Engineering, Guangxi Normal University, Guilin, 541004, China
[c] School of Engineering and Technology, University of Washington Tacoma, Tacoma, WA 98402, USA
[d] Department of Electrical/Computer Engineering, College of Engineering and Computer Science, The University of Texas Rio Grande Valley, Edinburg, TX 78541, USA
[e] Laboratory of Computer Network and Information Integration, Southeast University, Nanjing, China

ARTICLE INFO

Keywords:
Mobile edge computing
Deep reinforcement learning
Computation offloading
Delay
Energy consumption

ABSTRACT

Mobile edge computing (MEC) is an effective computing paradigm for mobile devices in the 5G era to reduce computing delay and energy consumption. However, in a multi-user resource competition environment, the revenue-driven behavior of edge servers will cause some users to increase delays or fail tasks. Considering this situation, we take the success rate of computation offloading as the trust value of the edge server, and build a system model from the user's perspective, taking delay and energy consumption as the multi-objective task of joint optimization. In the optimization goal, we consider three factors: offloading delay, energy consumption, and queuing delay. Simultaneously minimizing energy consumption and delay is a contradiction problem. Therefore, we solve the problem based on the principle of reducing energy consumption as much as possible when the offload success rate (decreasing delay) is prioritized. Further, we build the problem as a Markov decision problem (MDP) with multi-factor reward value, and treat the trust value as a state of the system. Finally, we use an extended deep deterministic policy gradient (DDPG) algorithm (a DDPG algorithm with multi-objective reward) to work around this problem. Experimental results show that our proposed scheme can better reduce the delay and energy consumption in computation offloading of mobile users (MUs) significantly better than the baseline schemes. The advantages of our proposed scheme are more obvious in an environment where computing resources are tight.

1. Introduction

In recent years, with the popularization of mobile terminals such as smart phones and wearable smart devices, more and more new intensive computing applications are executed on mobile terminals, such as augmented reality (AR), virtual reality (VR), etc. However, due to the limitations of its own computing power and battery power, mobile users (MUs) cannot quickly execute these applications or work continuously for a long time. Therefore, service delays and energy consumption will affect the normal use of users. In order to address these bottlenecks, computation offloading has become an effective solution [1]. However, offloading computation task to the cloud server will bring high delay and bandwidth costs [2]. In order to process computation tasks in time, it is proposed to migrate servers to the edge of the network to provide computation offloading services for MUs to avoid high transmission delay, which is called mobile edge computing (MEC) [3].

The popularization of 5G networks has laid the foundation for the Internet of Everything, which will also spawn more and more computing-intensive mobile applications [4]. As an excellent solution for 5G networks, MEC has received extensive attention and research [5,6]. With the popularity of 5G, edge servers (ESs) will be deployed in places with high population flow (such as stadium, hospital, shopping mall, etc.) [7]. MUs and ESs are selfish and rational. Generally, the edge service is provided in a pay-as-you-go method, that is, the user pays the service fee for computation offloading to the edge service provider. Edge computing services has also become an area where many computing service providers compete [8]. In a real-world edge computing scenario, there are often multiple edge service providers [9]. From the service provider's point of view, each ES should provide more services in order to increase profit [10,11]. In the scenario of multi-user edge resource competition, the server has more initiative. In situations where a large number of offload tasks need to be queued, an unconstrained

* Corresponding author.
E-mail address: lixx@gxnu.edu.cn (X. Li).

https://doi.org/10.1016/j.comnet.2022.109181
Received 8 January 2022; Received in revised form 22 June 2022; Accepted 9 July 2022
Available online 18 July 2022

ES may prioritize larger data in order to quickly get more revenue. In this way, some MUs may suffer from unreasonable arrangements. Since cross-server migration can be more expensive [12], MUs typically do not deviate from assigned servers. Because of this, when the edge server does not process tasks according to the order of task submission in order to obtain more revenue quickly, the quality of service of some computation offloading tasks of MUs may not be guaranteed. Therefore, this paper pays attention to the problem of multi-user computation offloading in edge computing system with resource competition.

Recently, some studies have started to use deep reinforcement learning (DRL) approaches instead of traditional approaches to solve the problems of MEC systems [13–15]. Huang et al. [13] paid attention to the energy consumption of the MEC system, and proposed real-time reinforcement learning offloading scheme. They uses double Q-learning to perform computation offloading scheduling for reducing the total energy consumption of the system. Tang et al. [14] focused on the computation offloading problem of task indivisible delay constraints, using long short term memory (LSTM) and double deep-Q network (DDQN) technology to make offloading decisions, reducing the task failure rate and average delay. Huan et al. [15] studied the computation offloading and resource allocation of dynamic multi-user MEC systems, and used the DDQN algorithm to target delay constraints and resource requirements to minimize energy consumption. These studies are all considered from the perspective of service providers, assuming that ESs follow allocated resources. In fact, ESs belonging to multiple service providers are all about maximizing their own profits. Therefore, we build trust values for ESs from the user's point of view. Similar to some existing studies [16–19] inducing the device trust value from the interaction records of the system, we construct the trust value of the ES based on the success rate of computation offloading. Differing from these previous studies, we take the trust value as one of the states of the system environment, and use the agent to interact with the environment to make decisions to solve the multi-user computation offloading problem in the resource competition environment. In addition, differing from studies on the single-objective optimization problem [13–15] or multi-objective separate optimization problem [20], we regard the delay and energy of MEC system as a multi-objective problem of joint optimization. Specifically, we study a multi-objective joint optimization problem in a resource-competitive environment with multi-server and multi-user. The contribution of this article can be summarized as following:

1. This paper studies a resource competition computation offloading scenario with multi-user and multi-server. In order to ensure the user's service quality, we use the server trust value based on the offloading success rate as one of the system states to build a multi-objective Markov decision problem (MDP) model for reducing delay and energy consumption.
2. In order to adapt for the multi-factor reward MDP, we extend the reward of the traditional deep deterministic policy gradient (DDPG) algorithm to multi-dimensional, and propose a new computation offloading method based on the extended DDPG algorithm. Compared with the deep-Q network (DQN) algorithm, the DDPG algorithm can adapt to continuous action space and high-dimensional state space problems. In this paper, the extended DDPG algorithm is used to obtain the optimal strategy for user scheduling and resource allocation in the edge computing system with multi-user resource competition.
3. We conduct extensive simulation experiments. The scheme we proposed has lower delay and energy consumption than the baseline schemes. We measure the offloading delay and energy consumption of servers with different numbers of users and different computing capabilities and further demonstrate the superior performance of our proposed scheme. In addition, compared to greedy strategies with the offloading rate of 0 and the offloading rate of 1, our scheme can significantly reduce the average delay.

The rest of the paper is organized as follows. Section 2 overviews related work. Section 3 presents system model and formulation of the model. Section 4 proposes DRL-based computation offload algorithm. Section 5 shows extensive performance evaluation through a series of simulation experiments. Section 6 concludes the work.

2. Related work

Reducing delay and energy consumption has always been a hot topic in edge computing research. Zhou et al. [21] considered microservices in mobile edge computing environments, took minimizing delay as a constraint, and proposed a delay-aware approximate algorithm to reduce the consumption of network resources. Xu et al. [22] considered the delay tolerance of different offloading tasks and designed an efficient online algorithm to maximize the number of offloading requests within a limited allowable time. You et al. [23] studied the problem of multi-user MEC resource allocation based on time-division multiple access (TDMA) and orthogonal frequency-division multiple access (OFDMA), and transformed the resource allocation problem into a weighted convex optimization problem of minimum mobile energy consumption. According to the user's channel gain and the priority of locally calculated energy consumption, an optimal resource allocation strategy based on the threshold structure is constructed. Hu et al. [24] focused on minimizing the total transmission energy of the access point (AP) under a wireless powered MEC system, considering the "double-near-far" effect, and designed a low-complexity algorithm to solve the problem of minimizing the AP's transmit power.

The above studies have considered the specific constraint model and expect to obtain an approximate optimal solution. Since there are many uncertain behaviors of objects in the MEC environment, it has become popular to use machine learning to solve problems in the MEC environment. Compared with deep learning, DRL does not need to manually label samples, and it can adopt an automatic update strategy for state changes without manual participation. Li et al. [25] proposed a resource allocation framework for a multi-user wireless MEC system, taking the delay cost and energy consumption of the system as optimization goals, and solving it through a DQN-based solution. Liu et al. [26] designed an Internet of things (IoT) device offloading scheme for the MEC system, and solved the optimization problem of system delay and energy consumption through the Q-learning algorithm based on the greedy strategy. Xu et al. [27] proposed an efficient reinforcement learning-based resource management algorithm through value iteration and reinforcement learning decomposition to improve the service quality of renewable energy MEC systems and minimize system energy and delay costs. Min et al. [28] proposed a DQN-based algorithm to solve the computation offloading problem of IoT devices with energy harvesting functions in a dynamic MEC network to reduce energy consumption, computation delay and packet loss rate. Chu et al. [29] established a multi-user computation offloading model and proposed a computation offloading strategy based on DQN, which minimizes the weighted total overhead of delay and energy consumption as the optimization objective. Li et al. [30] studied a dynamic offloading problem of multi-user MEC networks and designed an offloading strategy based on DQN to ensure system performance with delay and energy consumption as optimization goals. Since DQN does not adapt to the problem of continuous action space, Chen et al. [31] proposed a mobile fog edge computing model, and chose to use DDPG to solve the problem of fog resource allocation in computation offloading. Wang et al. [32] proposed to use DDPG-based algorithms in the unmanned aerial vehicle assisted MEC system to reduce the delay of computation offloading. Xia et al. [33] used the deep Q-learning algorithm to solve the problem of location-aware 5G multi-unit mobile task offloading to reduce delay and energy consumption. Chen et al. [34] studied the computation offloading problem of the integration of AR and next generation IoT. They proposed a binary offloading scheme and used an improved DDPG algorithm to make offloading decisions to reduce delay. Dai et al. [35]

proposed a joint problem of resource allocation and computation offloading with delay constraints under the cloud-side-end collaborative network, and used the DDPG algorithm to reduce energy consumption. Liu et al. [36] paid attention to the delay and long-term energy consumption of user computation offloading in a multi-base station environment, and constructed a MDP to solve it using the DDQN-based algorithm. Zhu et al. [37] studied a resource management problem in vehicular edge computing (VEC), adopted an incentive mechanism to incentivize the VEC server to participate in the resource allocation process, and proposed a DRL based resource management scheme to maximize the profits of vehicles and VEC server. Chen et al. [38] studied a polling-callback energy-saving offloading strategy under a fully asynchronous offloading system, modeled the time-sharing MEC data transmission problem as the total energy consumption minimization model, and proposed an algorithm combining DDQN and distributed LSTM to improve the ability of processing and predicting time intervals and delays in time series. Wang et al. [39] designed a framework with multiple static and vehicle-assisted MEC servers to handle the workloads offloaded by wireless users, and adopted an offloading strategy based on the DDQN algorithm to optimize the system's energy and latency costs. Nduwayezu et al. [40] studied the joint offloading and resource blocks assignment problem in a multi-carrier nonorthogonal multiple access based MEC system, and developed a two-stage DRL algorithm to solve the problem of maximizing the sum computation rate while satisfying delay and energy consumption constraints. By analyzing the above research background, we can draw the following conclusions: (1) [25,26,28,31,33,40] studied the multi-objective optimization problem based on multi-user and single-server. (2) [32,34,37,38] studied the optimization problem based on multi-user and single-server. (3) [35] studied the single-objective optimization problem based on multi-user and multi-server. (4) [27,36,39] studied the multi-objective optimization problem based on multi-server (where the number of mobile devices is unknown). Although [29,30] study the multi-objective optimization problem based on multi-server and multi-user, this paper is different from them in that we study a multi-server and multi-user resource competition scenario, and take the offloading success rate as the server trust value as a state of the system. Furthermore, we treat energy and delay as a joint multi-objective optimization task, and adopt an extended DDPG algorithm to optimize the offloading strategy.

3. System model and problem formulation

In this section, we explain our system model and system communication model, delay and energy model, and the formulation of problem.

3.1. System model

This article studies a resource contention scenario with multi-user on multi-server. We assume that N ESs (the servers belong to multiple service providers) are deployed in a densely populated city center, where computing resources are usually tight. In the scenario, we assume that there is a management center that provides users with an application that assists in computation offloading. Note that for users who access the management center, they will accept the decisions provided by the management center. We consider an OFDMA-based scheme, where each edge server that provide computation offloading services deploys a base station. Using OFDMA in a wireless network can well suppress interference between devices [35,41]. As shown in Fig. 1, the management center collects brief offloading demand information (battery status, location, computation task data size) from MUs, and provides decision-making information for MUs according to the status of ESs (servers performance and status information is public). When MUs have sufficient energy and computation power, tasks can be computing locally. In order to reduce energy consumption and improve computation efficiency, ESs can also be used to perform computation offloading services. MUs offload the task data to the ESs through the base station. Similar to [36], we discretize time into a set of equal-interval time slots with a total set of time slots denoted as $\boldsymbol{T_s} = \{1...T_s\}$. The computation task of the MU i in the time slot k is denoted as C_i^k, $k \in \boldsymbol{T_s}$. Similar to [28], the computation task of the MU i can be divided into parts of different sizes, one part can be selected to be offloaded to the ES, and the other part can be placed locally for computation. $\varrho \in [0,1]$ is the offloading rate. Specifically, the MU i offloads the ϱC_i^k part of the computation task C_i^k to the ES for computation, and the remaining $(1-\varrho)C_i^k$ part is computed locally. When $\varrho = 0$, all tasks are computed locally of the MU, and when $\varrho = 1$, all tasks will be offloaded to the ES. f_i^k and f_j^k respectively represent the device CPU frequency of the MU i and the CPU frequency of the ES j when processing computation tasks. As with some work focusing on delay and energy [29,30,35,36], we adopt a widely used delay and energy model. Table 1 lists the commonly used symbols.

Table 1
Symbol description.

Symbol	Definitions
i	Represents the MU i
j	Represents the ES j
M	Number of MUs
N	Number of ESs
$e_j(x_j^0, y_j^0)$	Coordinates of ES j
$d_i(x_i^0, y_i^0)$	Initial coordinates of MU i
$d_i(x_i^1, y_i^1)$	The moved coordinates of MU j (coordinates of the next time slot).
D_{ij}	The absolute distance between the ES i and the MU j.
ϱ	Task offload rate
C_i^k	Offloading task of user i in time slot k
σ	Channel gain at a distance of one meter
r_{ij}^k	The transmission rate at which mobile user i offloads tasks to edge server j in time slot k.
ω	The link transmission bandwidth of uploaded data for per MU.
ϕ^2	Noise power
P_{ij}^t	Transmission power
$O_{ij} = 1$	Indicates that there is an obstacle
$O_{ij} = 0$	Indicates that there are no obstacles
Γ_{nlos}	Transmission loss
f_i^k	CPU frequency of mobile user i
f_j^k	CPU frequency of edge sever j
Y	The number of cycles required by the CPU to process per bit
l_q	Data size of the offload queue
κ	Impact factor of CPU architecture
ξ	Delay tolerance
η_1	Weight parameter of offloading delay
η_2	Weight parameter of energy consumption
η_3	Weight parameter of queuing delay

Remark. Similar to [42], we assume that MUs transmit offloading requirements to the management center and receive decision information through a dedicated channel, which does not affect the performance of MUs offloading data to ESs. The management center provides an application that serves MUs task offloading, and obtains the MUs' battery status and location information, the next task data, and the completion status of the MUs' previous task offloading ESs on this application, including the time of task upload and computation complete time. The computing power of the ESs and the size of the upload bandwidth allocated by the ESs are publicly available.

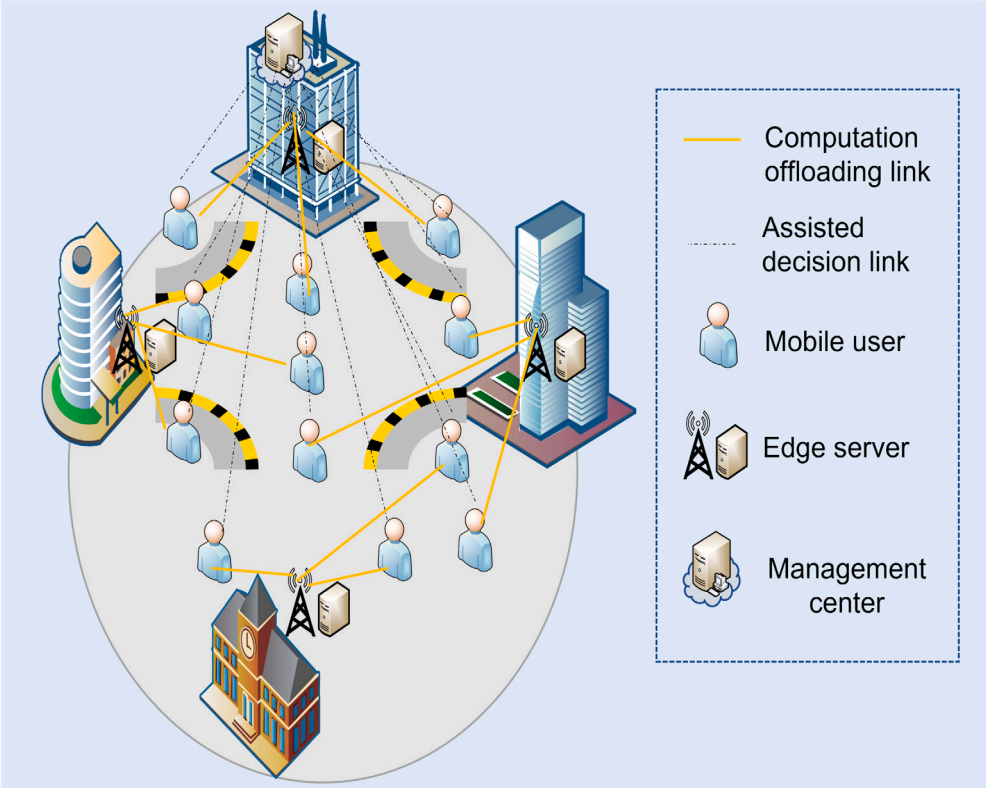


Fig. 1. MEC system computation offloading structure.

3.2. Mobile communication model

Considering that users have a certain mobility in reality, mobility also leads to changes in data transmission rates. Therefore, we refer to the mobility model of [32] to calculate the absolute distance of the MUs to the ESs for each time slot. Similarly, we assume that MUs are randomly walking in the area at low speed. Similar to [36], for convenience we assume that the user's location and communication conditions remain unchanged during each time slot. Each MU selects an ES to offload part of the data to the server for computing, and the rest is placed locally for computation. ESs are fixed locations. We assume that the coordinates of the ES j is $e_j(x_j^0, y_j^0)$. The starting coordinates of the MU i is $d_i(x_i^0, y_i^0)$, and the coordinate of the end point after moving is $d_i(x_i^1, y_i^1)$. Therefore, the channel gain of the link between the ES j and the MU i can be expressed as following [32]:

$$g_{ij} = \sigma D_{ij}^{-2} = \frac{\sigma}{|d_j(x_j^1, y_j^1) - e_i(x_i^0, y_i^0)|^2 + h^2} \tag{1}$$

Where σ represents the channel gain at a distance of one meter, D_{ij} is defined as the absolute distance between the ES i and the MU j, and h is the relative height of the base station. The relative height value is the base station antenna height minus the user antenna height. Considering that there may be obstacles (such as houses, trees, etc.) directly between the MU and the ES, this will affect the data transmission speed between the ES and the MU. According to [32,35,36], the data transmission rate r_{ij}^k between the ES j and the MU i is as following:

$$r_{ij}^k = \omega log_2(1 + \frac{P_{ij}^k g_{ij}}{\phi^2 + O_{ij}\Gamma_{nlos} + \sum_{i' \in M/\{i\}, j' \in N/\{j\}} g_{i'j'} P_{i'j'} (D_{i'j'})^{-\alpha}}) \tag{2}$$

Where ω is the link transmission bandwidth per MU uploading ES. ϕ^2 is defined as the noise power, and P_{ij}^k is the transmission power from the MU i to the ES j. O_{ij} defines the obstacles between the MU i and the ES j. When O_{ij} is 1, it means there are obstacles, and when O_{ij} is 0 it means there is no obstacle. Γ_{nlos} represents transmission loss. We assume MUs associated with the same ES are allocated orthogonal spectrum, so we only consider the interference among MUs associated with different ESs [43]. Similar to [35], $\sum_{i' \in M/\{i\}, j' \in N/\{j\}} g_{i'j'} P_{i'j'} (D_{i'j'})^{-\alpha}$ is the interference from other ESs. α is the path loss exponent.

3.3. Multi-user computation offloading model

In the computation model, we consider the calculation of task offloading delay, energy, and server trust value. Similar to [30,35] since the returned data result is much smaller than the offloaded data, we ignore the delay and energy consumption caused by returning the result.

3.3.1. Offloading delay calculation

In time slot k, MU i can offload part of data C_i^k to an edge server for computation, and the remaining $(1-\varrho)C_i^k$ data can be computed locally . According to [30,35], the delay for performing the computation locally is as following:

$$t_{local}^k = \frac{(1-\varrho)C_i^k Y}{f_i^k} \tag{3}$$

Y represents the CPU cycles required to compute 1 bit. The delay of offloading to the edge server can be divided into transmission delay and offloading computation delay. By formula (2), the transmission delay can be expressed as following:

$$t_r^k = \frac{C_i^k}{\omega log_2(1 + \frac{P_{ij}^k g_{ij}}{\phi^2 + O_{ij}\Gamma_{nlos} + \sum_{i' \in M/\{i\}, j' \in N/\{j\}} g_{i'j'} P_{i'j'} (D_{i'j'})^{-\alpha}})} = \frac{C_i^k}{r_{ij}^k} \tag{4}$$

The offloading computation delay of the ES j is divided into the time waiting at the server queue (i.e., queuing delay) and the computation execution time. The expression of queuing delay is as following:

$$t_{wait}^k = \frac{l_q Y}{f_j^k} \tag{5}$$

Where l_q is the data size of the ES j queue when offloading data is offloaded, and the execution computation time is as following:

$$t_e^k = \frac{\varrho C_i^k Y}{f_j^k} \tag{6}$$

Combining formula (5),(6) we can get the offloading computation delay expression as following:

$$t_{edge}^k = \frac{(l_q + \varrho C_i^k)Y}{f_j^k} \tag{7}$$

From the formula (4) and (7), the delay of offloading the computation to the edge server is as following:

$$t_{off}^k = \frac{C_i^k}{\omega log_2(1 + \frac{P_{ij}^k g_{ij}}{\phi^2 + O_{ij}\Gamma_{nlos} + \sum_{i' \in M/\{i\}, j' \in N/\{j\}} g_{i'j'} P_{i'j'} (D_{i'j'})^{-\alpha}})} + \frac{(l_q + \varrho C_i^k)Y}{f_j^k} = t_r^k + t_{edge}^k \tag{8}$$

Since the data offloading computation part and the local execution computation part are carried out at the same time, the total delay of the computation offloading is expressed as following:

$$T_{total}^k = max\{t_{local}^k, t_{off}^k\} \tag{9}$$

3.3.2. Energy consumption calculation

From users' perspective, we only consider the energy consumption of MU. When a MU performs computation offloading, its energy consumption can be divided into the energy consumption of local computing and the energy consumption of transmission during the offloading process. The local execution energy consumption of the MU is related to the CPU frequency and data size. According to [30,35], the energy consumption of the local execution data is as following:

$$E_{local}^k = \kappa(1 - \varrho) C_i^k Y (f_i^k)^2 \tag{10}$$

The size of κ depends on the capacitance coefficient of the CPU architecture. The transmission energy consumption of MU is related to transmission power and transmission time. Introducing formula (4), the energy consumption of mobile devices offloading data to the server is as following:

$$E_r^k = \frac{P_{ij} C_i^k}{\omega log_2(1 + \frac{P_{ij}^k g_{ij}}{\phi^2 + O_{ij}\Gamma_{nlos} + \sum_{i' \in M/\{i\}, j' \in N/\{j\}} g_{i'j'} P_{i'j'} (D_{i'j'})^{-\alpha}})} = P_{ij}^k t_r^k \tag{11}$$

During task offloading, MUs may be idle [44]. In this case, MUs need to wait until the ESs computing is complete. MUs' idle time will also generate energy consumption. In our environment, MUs and ESs compute simultaneously. There are two cases where the ES completes the computation first and the MU completes the computation first. When the ES completes the computation first, the MU i is idle and $t_{off}^k > t_{local}^k$. The idle time of MU i is $t_{off}^k - t_{local}^k$. Therefore, the energy consumption of MU i idle time is as following:

$$E_{idle} = P_x(t_{off}^k - t_{local}) \tag{12}$$

P_x represents idle power. Combining formulas (10) (11) and (12), the total energy consumption of mobile user for data offloading services is expressed as following:

$$E_{total}^k = E_{local}^k + E_r^k + E_{idle} \tag{13}$$

However, in the scenario where the energy consumption of the received data needs to be considered, the energy model can be easily extended by adding the energy consumption of receiving data to the formula (13).

3.3.3. Trust value calculation

In an environment of resource competition, when the ES wants to quickly obtain greater benefits, the ES may follow the order of data processing in tasks from large to small, rather than the order of task submission time. In this case, the quality of service for MUs with small data tasks will be low, and it may even lead to the failure of computation offloading. In view of the revenue-driven behavior of the server, we add the concept of trust values for the ESs to provide a basis for MUs computation offloading decisions. The trust values of the ESs are determined by the success rate of offloading tasks. For the success or failure of the offloading task, we make the following definitions: When the real total delay $D_{real} \leqslant min\{D_{local}, \xi T_{total}^k\}$, the task of offloading task is successful. $\xi > 1$ is the delay tolerance, and D_{local} represents the delay when all tasks are executed locally. We assume that the user can tolerate a certain amount of delay, and the maximum tolerable delay is equal to the delay when all tasks are computed locally. In reality, in order to reduce power consumption, users can accept that the offloading computation delay is equal to the delay computed locally. The trust value for the ES is calculated as following:

$$\Psi = \frac{F_j}{Z_j^{total}} \tag{14}$$

Where F_j is the number of successful offloading computations of ES j, and Z_j^{total} is the number of offloading computations by ES j. Under the same conditions, MUs will be preferentially select ESs with high trust values for offloading computations. Therefore, in the case of low trust value, the ES should pay more attention to the success rate rather than immediate profit to ensure its long-term interests.

3.4. Problem formulation

In fact, minimizing the energy consumption of MUs and minimizing the offloading delay is a contradictory problem. Based on the above model, we pay attention to the tradeoff between delay and energy, and ensure that the delay cannot exceed the maximum tolerable delay of the user when performing the offloading task, and also reduce the energy consumption of the MU as much as possible. In addition, ESs are not executed in the order of submission as shown in Fig. 2, which will cause a lot of wait time. Obviously, when the ESs prioritize processing large data tasks, the queuing delay of the system will increase. Therefore, queuing delay is an important factor in measuring the success of offloading computations. We take the weighted sum of offloading delay, energy consumption, and queuing delay as the joint optimization objective when considering server trust value, user location in each time slot, and obstruction, in order to optimize offloading decisions for reducing total delay and energy consumption. Our final optimization goal is expressed as following:

$$P_1 : G_{object} = min \frac{1}{T_s} \sum_{k=1}^{T_s} \sum_{i=1}^{M} \eta_1 T_{i \to total}^k + \eta_2 E_{i \to total}^k + \eta_3 t_{i \to total}^k \tag{15}$$

Where $\eta_1, \eta_2, \eta_3 \in [0, 1]$ and $\eta_1 + \eta_2 + \eta_3 = 1$. $T_{i \to total}^k$ represents the total delay for MU i to complete the offloading computation task in time slot k, $E_{i \to total}^k$ represents the energy consumed by MU i in completing the computation task of time slot k, and $t_{i \to total}^k$ represents the time that MU i needs to wait in the queue when offloading tasks in time slot k.

4. DRL for computation offloading

In this section, we describe our model for the computation offloading problem as a MDP with multi-objective rewards and define the state space, action space, and reward function in detail. We also present a computation offloading scheme based on an extended DDPG and the implementation algorithm.

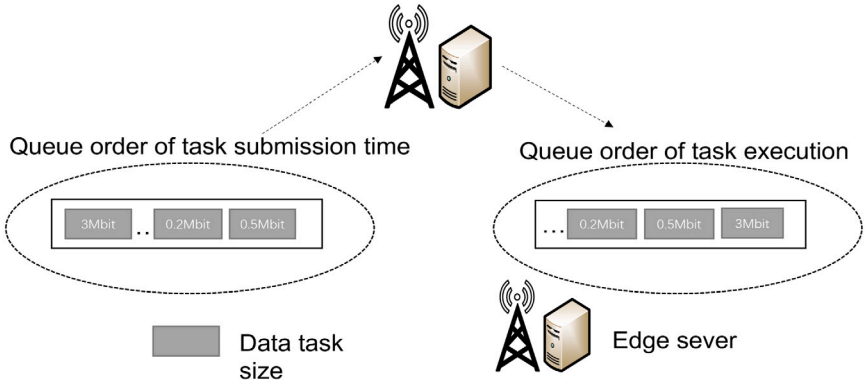


Fig. 2. Edge servers prioritize larger data tasks.

4.1. MDP modeling

According to the mode discussed in the previous section, each mobile user selects action a in the state s by relying on the strategy π in time slot k. To construct a MDP model, three basic elements, State Space, Action Space, and Reward Function, are included.

4.1.1. State space

$S = \{B_i, D_i^{location}, C_{size}^{i\to k}, \Psi_j, r_{ij}^k, \Gamma_{i\to nlos}, \ldots, D_M^{location}, B_M, C_{size}^{M\to k}, \Psi_N, r_{MN}^k, \Gamma_{M\to nlos}\}$.

B_i represents the battery status of MU i at the time slot k, $D_i^{location}$ represents the location of MU i, $C_{size}^{i\to k}$ represents the size of the data processed by MU i at the time slot k, Ψ_j represents the trust value of ES j, r_{ij}^k represents the data transmission rate from the ES j to the MU i at the time slot k, $\Gamma_{i\to nlos}$ represents the obstacles when MU i chooses to offload a task to the ES, and $\Gamma_{i\to nlos} = 1$ represents there are obstacles. $\Gamma_{i\to nlos} = 0$ means there are no obstacles, where $i \in \{1, 2...M\}$, and $j \in \{1, 2...N\}$.

4.1.2. Action space

$A = \{\Xi_{i\to j}^{id}, X_{i\to j}^{off} \ldots, \Xi_{M\to N}^{id}, X_{M\to N}^{off}\}$

$\Xi_{i\to j}^{id}$ represents the ID of the ES j that MU i chooses to offload, and $X_{i\to j}^{off}$ represents the portion of the data that MU i chooses to offload on the ES j. At time slot k, after an action A_k is taken, the system computes an immediate reward R before updating state S_k to the next state S_{k+1}.

4.1.3. Reward function

The ultimate goal of the MDP is to have an optimal decision-making action A for each state S to minimize offloading delay and energy consumption. Therefore, different from a single reward for a single objective optimization problem, we generalize the multiple rewards for multiple objectives into an objective function with parameters. The reward function is closely related to the objective function and constraint conditions discussed in the previous section. We formulate the reward function as following:

$R = -\varsigma[\eta_1 T_{i\to total}^k(s,a) + \eta_2 E_{i\to total}^k(s,a) + \eta_3 t_{i\to total}^k(s,a)]$

ς is to adjust the value of reward close to -1, which is more conducive to algorithm learning.

4.2. DRL-based computation offloading algorithm

Reinforcement learning is one of the traditional methods for solving the MDP. Due to the low efficiency of traditional reinforcement learning algorithms, the current reinforcement learning based on the deep neural network (DNN) adopts the method of constant approximation to improve efficiency. It has become more and more popular in dealing with complex MDP. Here we give a brief introduction to the Q-learning algorithm. Basically, its process is based on the state S_k, using the greedy method to select the action A_k, and enter the next state S_{k+1}, and get the reward R_k, use $\langle S, A, R, S'\rangle$ to update the Q table.

When updating the Q table, the greedy strategy is used to select A', $A' = max_a' Q(S', a')$ based on the state S_{k+1}. The action a is selected to maximize the value of Q as A' to update the value function. Since the Q table needs to store all the action values of each state, it will be extremely difficult to store and search when a large number of continuous states occur. DQN uses a neural network to replace the Q table, and solves the problem of excessive storage and excessive searching due to the excessive number of states in the continuous state space. The objective function is $y_k = r + \gamma max_a' Q^*(s', a'|\theta)$, that is, the

Algorithm 1 Computation offloading scheme based on the extended DDPG

Initialize:
Randomly initialize Actor network parameters θ^μ and Critic network parameters θ^Q;
Initialize Actor target network parameters $\theta^{\mu'}$ and target value network $\theta^{Q'}$;
Initialize experience replay pool R_p;
for each episode e_o=1 to e_{max} **do**
  Reset parameters of multi-user edge computing offload environment;
  Initial state random noise $\mathcal{N}$ and ;
  Randomly generate initial state s_i for each user $i \in M$ and extract its feature vector $\varpi(s)$;
  for each time slot $k = 1$ to T_s **do**
    Select action $a_k = \mu(\varpi(s)|\theta^\mu) + \mathcal{N}$ according to the policy of Actor network and the exploration noise;
    Perform action a_k, get reward $\boldsymbol{r}_k$ and the next state s_{k+1} from the environment;
    Store $(\varpi(s_k), a_k, \boldsymbol{r}_k, \varpi(s_{k+1}))$ in the experience replay pool R_p;
    Randomly sample a small batch of Ω experiences $(\varpi(s_\upsilon), a_\upsilon, r_\upsilon, \varpi(s_{\upsilon+1}))$ from R_p;
    Computing y_υ, $y_\upsilon = \varsigma\boldsymbol{\eta}^T\boldsymbol{r} + \gamma Q'(s_{\upsilon+1}, a_{\upsilon+1}|\theta^{Q'})$;
    Use the minimized loss function $J(\theta^Q) = \frac{1}{\Omega}\sum_{\upsilon=1}^{m}(y_\upsilon - Q(\varpi(s_\upsilon), a_\upsilon|\theta^Q))^2$ to update all the parameters of the Critic network θ^Q.;
    Update all the parameters of the Actor network θ^μ using the sampled policy gradient:
    $J(\theta^\mu) = \frac{1}{\Omega}\sum_{\upsilon=1}^{\Omega}\bigtriangledown_a Q(s_\upsilon, a_\upsilon|\theta^Q)|s = s_\upsilon, a = \mu(s|\theta^Q)|\bigtriangledown\theta^\mu\mu(s|\theta^\mu)|s = s_\upsilon$;
    Update the target network with a soft update strategy:
    $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$;
  end for
end for

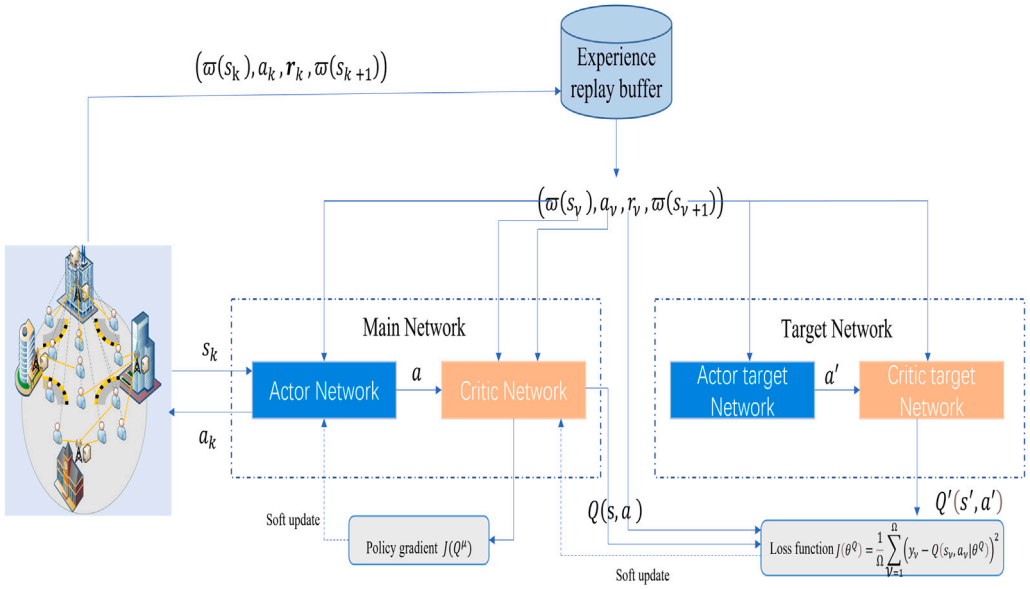


Fig. 3. Overall schematic framework diagram based on the extended DDPG.

next state uses the max function to select an action Q that maximizes the objective function, and γ is the discount factor. Since it is necessary to calculate the Q value of all actions, it becomes extremely difficult to exhaust all possibilities when the action space is continuous. Therefore, DQN cannot directly deal with the problem of continuous action space. DDPG overcomes the shortcomings of DQN's inability to adapt to the continuous action space.

Based on the above, we propose a computation offloading algorithm based on an extended DDPG. The DDPG separates the exploration of the action strategy from the learning update of the action strategy, and only changes the strategy to be learned to a deterministic strategy. By borrowing the actor–critic architecture, the policy network and the value network are separated, and experience reuse of DQN is used for non-policy training to minimize the relationship between samples. DDPG also uses normalized batch processing to prevent gradient explosions. DDPG defines deterministic behavior strategy μ, and each step of action can be calculated by $a_k = \mu(s_k)$. DDPG uses a DNN to simulate μ to become a strategy network with a parameter of θ^{μ}, and uses another DNN to simulate the Q function to become a value network with a parameter of θ^{Q}. Silver [45] proved that μ's gradient strategy is equivalent to the Q function gradient strategy, and its calculation formula is as following:

$$\nabla_{\theta^{\mu}} J = E_{\mu'}[\nabla_a Q(s_k, a_k|\theta^{Q})|s = s_k, a = \mu(s|\theta^{Q})|\nabla\theta^{\mu}\mu(s|\theta^{\mu})|s = s_k] \tag{16}$$

As shown in Fig. 3, the framework of the extend DDPG algorithm includes four networks, namely actor network, actor target network, critic network, and critic target network, respectively. Two actor network structures are the same, and the two critic network structures are the same. The actor network is responsible for the iterative update of the policy network parameters θ^{μ}, and selects the current action a_k according to the current state s_k, which is used to interact with the environment to generate the next state s_{k+1} and reward $\boldsymbol{r}$. And the multi-objective $\boldsymbol{r}$ is a three-dimensional vector, $\boldsymbol{r} = [T^k_{i\to total}, E^k_{i\to total}, t^k_{i\to total}]$. The actor target network is responsible for selecting the best next action a' based on the next state s' sampled in the experience replay pool. The network parameters $\theta^{\mu'}$ are regularly copied from θ^{μ}. The critic network is responsible for the iterative update of the value network parameters θ^{μ} and calculates the current Q value $Q(s, a|\theta^{Q})$. The calculated target Q value function is $y = \varsigma\boldsymbol{\eta}^T\boldsymbol{r} + \gamma Q'(s', a'|\theta^{Q'})$, $\boldsymbol{\eta} = [\eta_1, \eta_2, \eta_3]$. The critic target network is responsible for calculating the $Q'(s', a'|\theta^{Q'})$ part of the target Q value, and the network parameters $\theta^{Q'}$ are regularly copied from θ^{Q}.

The training method for the target network is different from that of the DQN. The soft update method is adopted during the training, and the formula of the soft update strategy is as following:

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \end{aligned} \tag{17}$$

τ is the update coefficient, $\tau \in [0, 1]$. For the randomness of the learning process and to explore potentially better learning strategies, noise $\mathcal{N}$ is added to action A, which is expressed as following:

$$A = \pi\mu(s) + \mathcal{N} \tag{18}$$

The loss function of critic network is similar to DQN, and the mean square error is used. The expression of the loss function is as following:

$$J(\theta^{Q}) = \frac{1}{\Omega}\sum_{v=1}^{m}(y_v - Q(s_v, a_v|\theta^{Q}))^2 \tag{19}$$

The purpose of actor network is to hope that critic network will feedback a large Q value. When the obtained Q value is larger, the loss will be smaller, and when the Q value is smaller, the loss will be greater. Therefore, the loss gradient of the actor network can be expressed as following:

$$J(\theta^{\mu}) = \frac{1}{\Omega}\sum_{v=1}^{m}\nabla_a Q(s_v, a_v|\theta^{Q})|s = s_v, a = \mu(s|\theta^{Q})|\nabla\theta^{\mu}\mu(s|\theta^{\mu})|s = s_v \tag{20}$$

We summarize the process multi-user computation offloading as Algorithm 1. The complexity of the algorithm is related to the structure of the deep neural network. The computational complexity of deep neural networks comes from the matrix calculation between layers and the calculation of activation functions at each layer. Therefore, the time complexity can be calculated as [46] $H = V_a * u_l + \sum_{l=1}^{l=L} u_l u_{l+1}$, where u_l means the unit number in the lth layer, V_a means the corresponding parameters are determined by the type of activation function. Suppose the actor network contains ϵ fully connected layers, and the critic network contains λ fully connected layers. Therefore, the time complexity of the actor network is H_{actor} and the time complexity of the critic network is H_{critic}. Since the main network and the target network have the same structure, the training complexity of the algorithm is estimated as $\boldsymbol{O}(e_{max} * T_s(H_{actor} + H_{critic}))$.

Table 2
Explanations of parameter default values.

Parameter	Definitions	Default values
C_i^k	The size of the data offloaded by MU i at time slot k	(1.5, 3) Mbit
Y	The number of cycles required by the CPU to process per bit	(900, 1100) cycles/bit
σ	Channel gain at a distance of one meter	−50 dB [32]
ω	The link transmission bandwidth of uploaded data for per MU	1 MHz
f_i^k	CPU frequency of MUs	1 GHz
f_j^k	CPU frequency of ESs	4 GHz
κ	Impact factor of CPU architecture	1e−27
h	Relative height of users and base stations	8.5 m
ξ	Delay tolerance	1.1
η_1	Weight parameter of offloading delay	0.6
η_2	Weight parameter of energy consumption	0.3
η_3	Weight parameter of queuing delay	0.1
ς	adjustable parameter	1e−5
A_lr	Actor network learning rate	10^{-4}
C_lr	Critic network learning rate	10^{-4}
γ	Discount factor	0.99
τ	Soft update factor	0.01

Table 3
η of different parameters values.

(η_1, η_2, η_3)	Average delay (s)	Average energy consumption (J)	Average offload rate
(0.7, 0.2, 0.1)	1.66	1.23	0.52
(0.65, 0.25, 0.1)	1.67	1.15	0.55
(0.6, 0.3, 0.1)	**1.65**	1.05	0.60
(0.6, 0.25, 0.15)	1.69	1.26	0.53
(0.55, 0.35, 0.1)	1.72	1.02	0.66
(0.4, 0.4, 0.2)	2.10	0.67	0.73

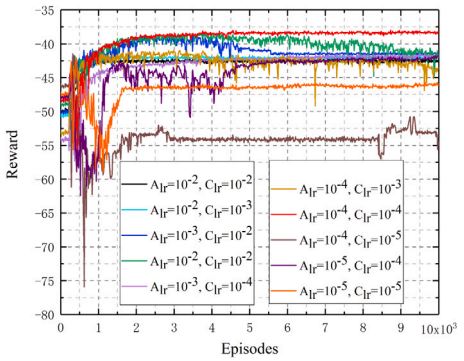


Fig. 4. Convergence performance of different learning rate.

5. Performance evaluation

In this section, we introduce simulation settings and baselines, and compare convergence performance, delay, and energy consumption from many aspects to evaluate the performance of our proposed scheme. The simulation experiment environment we used is TensorFlow 1.14.0 with python 3.6. The parameters set in the experiment and their default values are shown in Table 2.

5.1. Experimental settings

In order to verify the multi-server environment, we set up four ESs in an area of 300 × 300 m^2, the coordinates of which are fixed at [100,100], [100,200], [200,100], [200,200]. Similar to [35], we set 20 MUs randomly distributed in the area. In order to simulate the volatility of the computing process and tasks, the task data size and the required CPU cycles per bit of data are uniformly distributed similar to [47]. In order to reflect the mobility of MUs, we set its random movement speed to 1 m/s. Note that for the convenience of processing, we assume that the location of the mobile user does not change during each time slot. According to 3GPP TR 38.901 [48], the relative height of the base station is 8.5 m. The base station antenna height and user antenna height are 10 m and 1.5 m, respectively. The transmission power and idle power of UEs are 0.5 W and 0.1 W, respectively [44]. The link transmission bandwidth of per MU uploading data is set to 1 MHz [41]. The path loss exponent α is −2 [49]. The noise power adopts the 3GPP standard [35], the noise power is 10^{-11} mW, and the transmission loss is −20 dbm [50]. We have performed parameter tuning η many times. Under the requirement of delay, we firstly performed coarse-grained traversal of the value of η from 0.1–0.9 and then selected the value of η after fine-grained (0.05) traversal. When η_1 is around 0.6 and η_2 is around 0.3, the delay and energy consumption have reached a lower state (If η_1 is too large, the offload rate will decrease, and the local computation delay will be too large. If η_2 is too large, the offload rate will be too large, and the computing delay of offload to the server will increase.). η_3 mainly adjusts the queuing delay weight, and its value is small (The queuing delay is included in the delay model.). Such as the part results of a traversal in Table 3.

For the construction of DNNs, we choose two different networks. Actor has two hidden layers, and the number of neurons is 100 and 24, respectively. The critic network has three hidden layers, and the number of neurons is 200, 60, and 24, respectively. All hidden layers are fully connected and activated through the tanh function. We set the batch size to 64, and the experience reply pool size to 1e4. During the training process, we set the maximum number of episodes to 10 000, and the maximum time slot of each episode to 50.

Three baseline schemes we conducted experimental comparisons are as following:

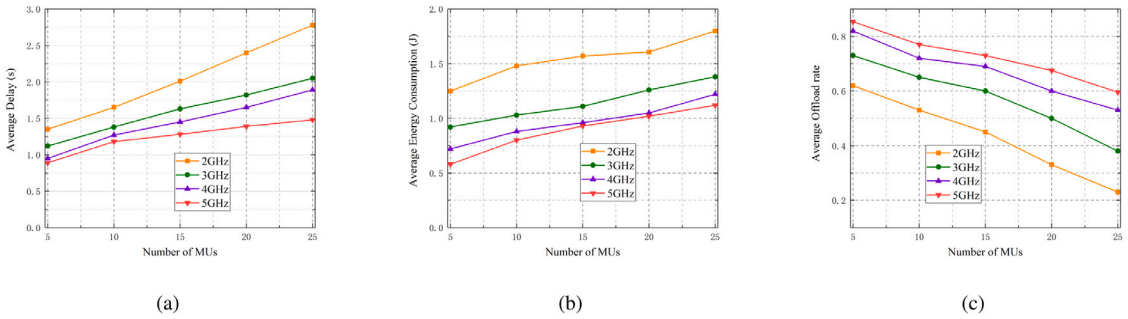
1. The offloading scheme based on DQN [30] algorithm.
2. The DDPG-based offloading scheme that does not consider the trust value of edge servers (No_trust).
3. The DDPG-based offloading scheme that consider the trust value of edge servers and does not add the wait time of mobile devices in the queue to the optimization goal (No_wait).

The DQN-based offloading scheme is designed using the discretization of the continuous action space. In addition, we test the effect of our proposed scheme with different parameters. We also compared the delays of offloading rates of 0 and 1 in greedy mode to test the performance of offloading schemes based on DRL algorithms.

5.2. Performance analysis

We evaluated the impact of different learning rates on the performance of the proposed algorithm. Fig. 4 shows the convergence trend of the learning rate A_{lr} of the actor network and the learning rate C_{lr} of the critic network from 10^{-2} to 10^{-5}. When $\{A_{lr} = 10^{-2}, C_{lr} = 10^{-2}\}$, $\{A_{lr} = 10^{-2}, C_{lr} = 10^{-3}\}$, $\{A_{lr} = 10^{-3}, C_{lr} = 10^{-4}\}$, $\{A_{lr} = 10^{-4}, C_{lr} = 10^{-4}\}$ and $\{A_{lr} = 10^{-5}, C_{lr} = 10^{-5}\}$, our proposed algorithm can converge faster. This converges to the maximum reward when $\{A_{lr} = 10^{-4}, C_{lr} = 10^{-4}\}$. When $\{A_{lr} = 10^{-3}, C_{lr} = 10^{-2}\}$ and $\{A_{lr} = 10^{-5}, C_{lr} = 10^{-4}\}$, the reward of our proposed algorithm can be slowly converged, but both fall into suboptimal solutions. When $\{A_{lr} = 10^{-2}, C_{lr} = 10^{-2}\}$, $\{A_{lr} = 10^{-4}, C_{lr} = 10^{-3}\}$, $\{A_{lr} = 10^{-4}, C_{lr} = 10^{-5}\}$, the reward of our proposed algorithm fluctuates greatly, cannot be converged.

Table 4 shows the average delay and average energy consumption when the number of MUs varies. As the number of MUs increases, the data task queue will increase, which will result in an increase for



(a) (b) (c)

Fig. 5. Average delay, average energy consumption, and average offload rate of different numbers of MUs when computation offloading under ESs with different CPU frequencies.

Table 4
The average delay and average energy consumption of task offloading with different numbers of MUs.

Average performance	M=5	M=10	M=15	M=20	M=25
Average delay (J)	0.95	1.27	1.45	1.65	1.89
Average energy consumption (s)	0.72	0.88	0.96	1.05	1.22
Average offload rate	0.82	0.72	0.69	0.6	0.53

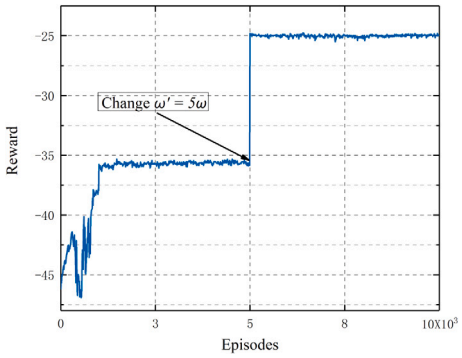


Fig. 6. Convergence performance of time-varying bandwidth.

queuing delay and idle energy consumption. On the other hand, as the number of users increases, the offloading rate of mobile user tasks will decrease, which will also increase the delay and energy consumption of MUs.

Fig. 5 shows the average delay and energy consumption of different numbers of MUs computation offloading under different CPU frequencies of the ESs. As shown in Fig. 5(a), when the number of MUs remains the same, as the computing power of the ESs increases, the average delay of MUs computation offloading decreases. When the computing frequency of the ESs are fixed, as the number of MUs increases, the average delay of computation offloading increases. As shown in Fig. 5(b), when the computing frequency of the ESs are fixed, the number of MUs increases, and the average energy consumption of MUs is increasing. When the number of MUs is fixed and the computing power of the ESs increases, the average energy consumption is decreasing. The change in average delay and energy consumption is largely due to the MUs changing the offload rate. Fig. 5(c) can confirm this point. As shown in Fig. 5(c), when the ESs' CPU frequency is fixed, as the number of MUs increases, the offload rate of MUs decreases. This is to avoid excessive waiting, and MUs reduce the offload rate. On the whole, our proposed scheme can get good performance.

To quickly evaluate the effect of time-varying bandwidth for the proposed algorithm, we change the environment to ten MUs and four ESs. After running the algorithm to 5000 episodes, we expand the bandwidth by five times. Fig. 6 shows that the proposed algorithm runs to around 1000 episodes and the reward value converges to about −36. At 5000 episodes, the bandwidth is changed to 5MHz, at which time the reward value increases sharply to around −25 and continues to converge stably. The experiment results have shown that our proposed algorithm can automatically adjust the policy and quickly converge to a new optimal solution in a time-varying bandwidth environment. In other words, our proposed algorithm scheme can properly adapt to the time-varying bandwidth environment.

5.3. Performance comparison

Fig. 7 shows the delay, energy consumption, and offload rate of our proposed scheme and the baseline scheme under different numbers of MUs. Fig. 7(a) shows that our proposed scheme is significantly better than the DQN-based scheme for reducing delay. The scheme that considers the trust value has a better effect on reducing the average delay than the No_trust scheme, and this will become more obvious as the number of MUs increases. Since the reward function adds queuing delay, our proposed scheme can reduce the average delay better than the No_wait scheme. In terms of energy consumption, as shown in Fig. 7(b), the average energy consumption of the DDPG-based scheme is less than that of the DQN-based scheme, and our proposed scheme is also slightly better than the No_trust and No_wait schemes. Fig. 7(c) shows the changes in the offload rate of different numbers of MUs. Our proposed scheme has a higher offload rate compared with the baseline scheme. As the number of MUs increases and the computing resources are tighter, the advantages of our proposed scheme are more evident.

Fig. 8 shows the average delay, energy consumption, and offload rate of ESs with different CPU frequencies compared with the baseline schemes. As shown in Fig. 8, as the computing powers of the ESs increases, the average delay and energy consumption are decreasing, and the corresponding MUs offload rate is increasing. As shown in Fig. 8(a) and (b), our proposed scheme has more obvious advantages in reducing delay and energy consumption when computing resources are tight. With the increase of the computing powers of the ESs, the performance advantage of the proposed scheme decreases with that of the No_wait scheme and the No_trust scheme, but it is still significantly better than the DQN-based scheme. This is also reflected in the trend in Fig. 8(c).

We compare the delays based on the greedy strategy with an offload rate of 0 (completely computed locally) and an offload rate of 1 (completely offloaded to the edge server). (Since we only focus on the energy consumption of MUs, there is no comparative value in the case of an offload rate of 1 with an energy consumption of 0, so we only compared the delay situation.) Fig. 9 shows the average delay of the offloading scheme based on the deep reinforcement learning algorithm and the offload rate based on the greedy strategy of 0 and 1, respectively. As the number of MUs increases, there will be an increase in queue tasks. Therefore, the average delay of completely offloading to the edge server becomes greater. As can be seen overall, the computation offloading scheme based on deep reinforcement learning has lower delay than the offloading rate of 0 or 1 under the greedy strategy, and our proposed scheme is better than the DQN-based scheme.

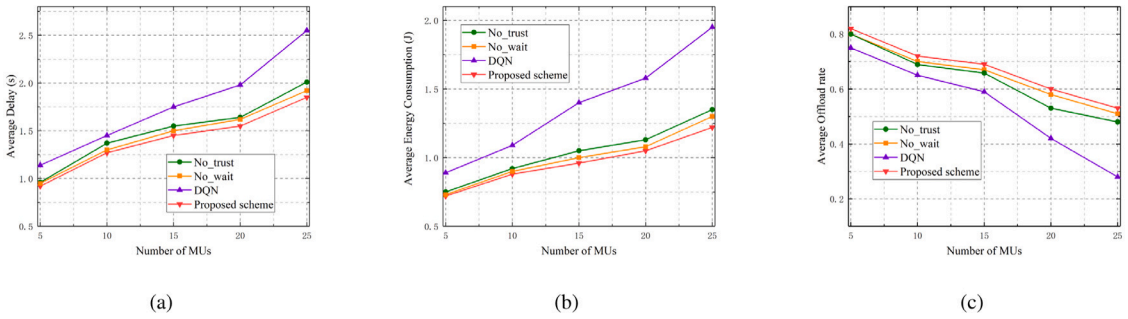


Fig. 7. Average delay, average energy consumption, and average offload rate compared to the baseline schemes under different numbers of MUs.

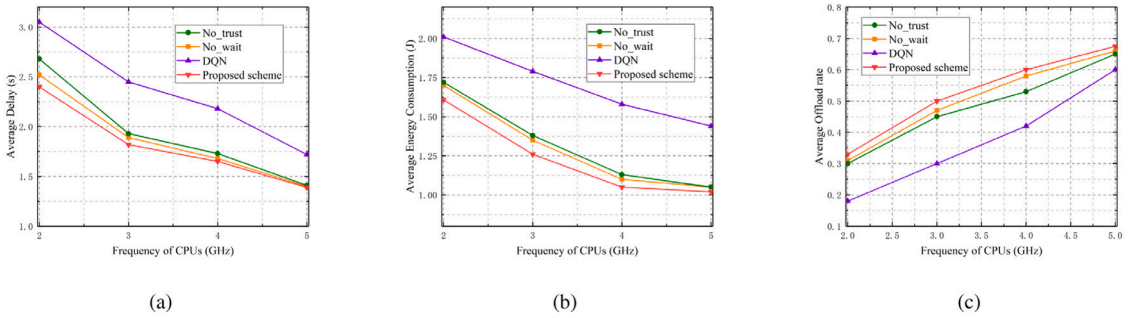


Fig. 8. Average delay, average energy consumption, and average offload rate compared to the baseline schemes under ESs with different CPU frequencies.

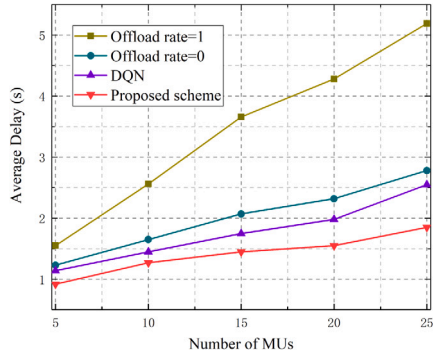


Fig. 9. Compared with the average delay of offload rate of 0 and 1, respectively.

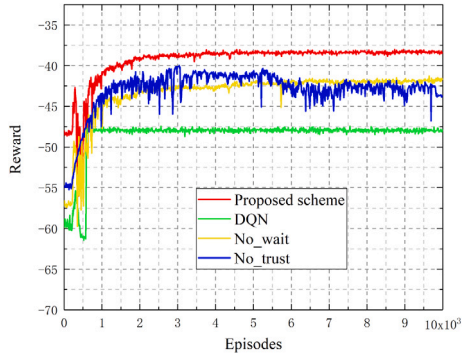


Fig. 10. Performance of offloading algorithm compared with baseline.

In order to verify the effectiveness of our proposed scheme, we compared the training performance of the baseline scheme. As shown in Fig. 10, our proposed scheme can get a higher reward value. The convergence effect of our proposed scheme is obviously better than the baseline schemes. DQN-based scheme can converge quickly, but falls into a sub-optimal solution. In the high-complexity state space and continuous action space scenarios, our proposed scheme is obviously better than the DQN-based scheme. Comparing the No_wait and No_trust schemes, our proposed scheme can converge to the optimal solution more quickly. Since the No_trust scheme does not consider the trust values of ESs, it has the worst convergence effect in an environment of resource competition.

6. Conclusion

In this paper, we consider the problem of multi-user computation offloading in an environment where edge computing resources are competitive. We propose to establish the trust values of the ESs based on the success rate of offload, and regard reducing energy and delay as a multi-objective optimization problem. Then, we build a MDP model with offloading delay, energy consumption and queuing delay as multiple reward factors, and use an extended DDPG algorithm to solve it. We execute a series of experiments to verify the effectiveness of the scheme. The results show that our proposed scheme can reduce delay and energy consumption better than the baseline scheme in a multi-user resource competition environment. However, this article still has some shortcomings, such as not considering the price model of ESs computing resources. In future work, we intend to study the pricing model of server computation offloading in a resource-competitive environment.

CRediT authorship contribution statement

Bin Qu: Investigation, Methodology, Resources, Visualization, Software, Writing – original draft, Writing – review & editing. **Yan Bai:** Investigation, Writing – review & editing, Supervision. **Yul Chu:** Investigation, review & editing, Supervision. **Li-e Wang:** Investigation, Writing – review & editing. **Feng Yu:** English editing and correction. **Xianxian Li:** Project administration, Validation, Investigation, Supervision, Funding acquisition, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This work was supported in part by the Guangxi "Bagui Scholar" Teams for Innovation and Research Project, China, in part by the Guangxi Collaborative Innovation Center of Multi-source Information Integration and Intelligent Processing, China, in part by the Guangxi Talent Highland Project of Big Data Intelligence and Application, China, the Guangxi Natural Science Foundation (Nos. 2020GXNSFAA297075, 2018JJA170082 and 2019JJA170060), in part by the Research Fund of Guangxi Key Lab of Multi-source Information Mining& Security (No. 19-A-02-02), in part by the National Science Foundation, USA Grant (No. 1921576), in part by the National Natural Science Foundation of China under Grants (No. 62062016), in part by the Key Laboratory of Computer Network and Information Integration (Southeast University), China, Ministry of Education-supported Project (No. K93-9-2020-04), and in part by the Innovation Project of Guangxi Graduate Education, China (No. JXXYYJSCXXM-2021-014).

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646.
[2] D. Huang, P. Wang, D. Niyato, A dynamic offloading algorithm for mobile computing, IEEE Trans. Wirel. Commun. 11 (6) (2012) 1991–1995.
[3] M.T. Beck, M. Werner, S. Feld, Mobile edge computing: A taxonomy, in: Proc. of the Sixth International Conference on Advances in Future Internet, Citeseer, 2014, pp. 48–55.
[4] N. Abbas, Y. Zhang, A. Taherkordi, T. Skeie, Mobile edge computing: A survey, IEEE Internet Things J. 5 (1) (2018) 450–465.
[5] T.X. Tran, A. Hajisami, P. Pandey, D. Pompili, Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges, IEEE Commun. Mag. 55 (4) (2017) 54–61.
[6] Y. Yu, Mobile edge computing towards 5G: Vision, recent progress, and open challenges, China Commun. 13 (Supplement2) (2016) 89–99.
[7] ETSI MEC ISG, Mobile edge computing-introductory technical white paper, 2014.
[8] T. Zhao, Z. Sheng, X. Guo, Z. Yun, Z. Niu, Pricing policy and computational resource provisioning for delay-aware mobile edge computing, in: Proc. of the IEEE/CIC International Conference on Communications in China, 2016.
[9] R.L. Aguiar, A. Sarma, D. Bijwaard, L. Marchetti, P. Pacyna, R. Pascotto, Pervasiveness in a competitive multi-operator environment: the daidalos project, IEEE Commun. Mag. 45 (10) (2007) 22–26.
[10] G. Cui, Q. He, F. Chen, Y. Zhang, H. Jin, Y. Yang, Interference-aware game-theoretic device allocation for mobile edge computing, IEEE Trans. Mob. Comput. (2021) http://dx.doi.org/10.1109/TMC.2021.3064063.
[11] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, Y. Yang, A game-theoretical approach for user allocation in edge computing environment, IEEE Trans. Parallel Distrib. Syst. 31 (3) (2019) 515–529.
[12] A. Mukhopadhyay, M. Ruffini, Learning automata for multi-access edge computing server allocation with minimal service migration, in: IEEE International Conference on Communications (ICC), IEEE, 2020, pp. 1–6.
[13] H. Huang, Q. Ye, H. Du, Reinforcement learning based offloading for realtime applications in mobile edge computing, in: Proc. of the IEEE International Conference on Communications (ICC), IEEE, 2020, pp. 1–6.
[14] M. Tang, V.W. Wong, Deep reinforcement learning for task offloading in mobile edge computing systems, IEEE Trans. Mob. Comput. (2020) http://dx.doi.org/10.1109/TMC.2020.3036871.
[15] H. Zhou, K. Jiang, X. Liu, X. Li, V.C.M. Leung, Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing, IEEE Internet Things J. 9 (2) (2022) 1517–1530.
[16] M.B. Mansour, T. Abdelkader, M. Hashem, E.-S.M. El-Horbaty, An integrated three-tier trust management framework in mobile edge computing using fuzzy logic, PeerJ. Comput. Sci. 7 (2021) e700.
[17] J. Guo, H. Wang, W. Liu, G. Huang, J. Gui, S. Zhang, A lightweight verifiable trust based data collection approach for sensor–cloud systems, J. Syst. Architect. 119 (2021) 102219.
[18] J. Liang, W. Liu, N.N. Xiong, A. Liu, S. Zhang, An intelligent and trust UAV-assisted code dissemination 5G system for industrial Internet-of-Things, IEEE Trans. Industr. Inform. 18 (4) (2021) 2877–2889.
[19] W. Kong, X. Li, L. Hou, J. Yuan, Y. Gao, S. Yu, A reliable and efficient task offloading strategy based on multi-feedback trust mechanism for IoT edge computing, IEEE Internet Things J. (2022) http://dx.doi.org/10.1109/JIOT.2022.3143572.
[20] Y. Wang, M. Sheng, X. Wang, L. Wang, J. Li, Mobile-edge computing: Partial computation offloading using dynamic voltage scaling, IEEE Trans. Commun. 64 (10) (2016) 4268–4282.
[21] A. Zhou, S. Wang, S. Wan, LMM: latency-aware micro-service mashup in mobile edge computing environment, Neural Comput. Appl. 32 (19) (2020) 15411–15425.
[22] Z. Xu, W. Liang, M. Jia, Task offloading with network function requirements in a mobile edge-cloud network, IEEE Trans. Mob. Comput. 18 (11) (2018) 2672–2685.
[23] C. You, K. Huang, H. Chae, Energy-efficient resource allocation for mobile-edge computation offloading, IEEE Trans. Wirel. Commun. 16 (3) (2016) 1397–1411.
[24] X. Hu, K.-K. Wong, K. Yang, Wireless powered cooperation-assisted mobile edge computing, IEEE Trans. Wirel. Commun. 17 (4) (2018) 2375–2388.
[25] J. Li, H. Gao, T. Lv, Deep reinforcement learning based computation offloading and resource allocation for MEC, in: Proc. of the IEEE Wireless Communications and Networking Conference, IEEE, 2018, pp. 1–6.
[26] X. Liu, Z. Qin, Y. Gao, Resource allocation for edge computing in iot networks via reinforcement learning, in: Proc. of the IEEE International Conference on Communications (ICC), IEEE, 2019, pp. 1–6.
[27] J. Xu, L. Chen, S. Ren, Online learning for offloading and autoscaling in energy harvesting mobile edge computing, IEEE Trans. Cogn. Commun. Netw. 3 (3) (2017) 361–373.
[28] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, W. Zhuang, Learning-based computation offloading for IoT devices with energy harvesting, IEEE Trans. Veh. Technol. 68 (2) (2019) 1930–1941.
[29] X. Chu, Z. Leng, Multiuser computing offload algorithm based on mobile edge computing in the internet of things environment, Wirel. Commun. Mob. Comput. 2022 (6107893) (2022) 9.
[30] C. Li, J. Xia, F. Liu, D. Li, L. Fan, G.K. Karagiannidis, A. Nallanathan, Dynamic offloading for multiuser muti-CAP MEC networks: A deep reinforcement learning approach, IEEE Trans. Veh. Technol. 70 (3) (2021) 2922–2927.
[31] M. Chen, T. Wang, S. Zhang, A. Liu, Deep reinforcement learning for computation offloading in mobile edge computing environment, Comput. Commun. 175 (2021) 1–12.
[32] Y. Wang, W. Fang, Y. Ding, N. Xiong, Computation offloading optimization for UAV-assisted mobile edge computing: a deep deterministic policy gradient approach, Wirel. Netw. 27 (4) (2021) 2991–3006.
[33] Q. Xia, Z. Lou, W. Xu, Z. Xu, Near-optimal and learning-driven task offloading in a 5G multi-cell mobile edge cloud, Comput. Netw. 176 (2020) 107276.
[34] M. Chen, W. Liu, T. Wang, A. Liu, Z. Zeng, Edge intelligence computing for mobile augmented reality with deep reinforcement learning approach, Comput. Netw. (2021) 108186.
[35] Y. Dai, K. Zhang, S. Maharjan, Y. Zhang, Edge intelligence for energy-efficient computation offloading and resource allocation in 5G beyond, IEEE Trans. Veh. Technol. 69 (10) (2020) 12175–12186.
[36] T. Liu, Y. Zhang, Y. Zhu, W. Tong, Y. Yang, Online computation offloading and resource scheduling in mobile-edge computing, IEEE Int. Things J. 8 (8) (2021) 6649–6664.
[37] X. Zhu, Y. Luo, A. Liu, N.N. Xiong, M. Dong, S. Zhang, A deep reinforcement learning-based resource management game in vehicular edge computing, IEEE Trans. Intell. Transp. Syst. 23 (3) (2022) 2422–2433.
[38] M. Chen, W. Liu, T. Wang, S. Zhang, A. Liu, A game-based deep reinforcement learning approach for energy-efficient computation in MEC systems, Knowl. Based Syst. 235 (2022) 107660.
[39] J. Wang, H. Ke, X. Liu, H. Wang, Optimization for computational offloading in multi-access edge computing: A deep reinforcement learning scheme, Comput. Netw. (2022) 108690.
[40] M. Nduwayezu, J.-H. Yun, Latency and energy aware rate maximization in MC-NOMA-based multi-access edge computing: A two-stage deep reinforcement learning approach, Comput. Netw. 207 (2022) 108834.
[41] C. Liu, K. Li, J. Liang, K. Li, COOPER-MATCH: Job offloading with a cooperative game for guaranteeing strict deadlines in MEC, IEEE Trans. Mob. Comput. (2019) http://dx.doi.org/10.1109/TMC.2019.2921713.
[42] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, L. Li, Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning, IEEE Trans. Cogn. Commun. Netw. 7 (3) (2021) 881–892.
[43] Y. Dai, D. Xu, S. Maharjan, Y. Zhang, Joint computation offloading and user association in multi-task mobile edge computing, IEEE Trans. Veh. Technol. 67 (12) (2018) 12313–12325.

[44] M.S. Hossain, C.I. Nwakanma, J.M. Lee, D.-S. Kim, Edge computational task offloading scheme using reinforcement learning for IIoT scenario, ICT Express 6 (4) (2020) 291–299.
[45] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M. Riedmiller, Deterministic policy gradient algorithms, in: Proc. of the International Conference on Machine Learning, PMLR, 2014, pp. 387–395.
[46] C. Qiu, Y. Hu, Y. Chen, B. Zeng, Deep deterministic policy gradient (DDPG)-based energy harvesting wireless communications, IEEE Internet Things J. 6 (5) (2019) 8577–8588.
[47] C. You, K. Huang, H. Chae, B.-H. Kim, Energy-efficient resource allocation for mobile-edge computation offloading, IEEE Trans. Wirel. Commun. 16 (3) (2016) 1397–1411.
[48] T.S. Rappaport, Y. Xing, G.R. MacCartney, A.F. Molisch, E. Mellios, J. Zhang, Overview of millimeter wave communications for fifth-generation (5G) wireless networks—With a focus on propagation models, IEEE Trans. Antennas Propag. 65 (12) (2017) 6213–6230.
[49] J. Miranda, R. Abrishambaf, T. Gomes, P. Gonçalves, J. Cabral, A. Tavares, J. Monteiro, Path loss exponent analysis in wireless sensor networks: Experimental evaluation, in: Proc. of 11th IEEE International Conference on Industrial Informatics, IEEE, 2013, pp. 54–58.
[50] M. Coldrey, J.-E. Berg, L. Manholm, C. Larsson, J. Hansryd, Non-line-of-sight small cell backhauling using microwave technology, IEEE Commun. Mag. 51 (9) (2013) 78–84.

Bin Qu is currently pursuing the Ph.D. degree at the College of Computer Science and Information Engineering, Guangxi Normal University, China. His research interests include edge computing, reinforcement learning, Internet of things, and blockchain system.

Yan Bai is a Professor in the School of Engineering and Technology, University of Washington Tacoma, USA. Dr. Bai received her Ph.D. in Electrical and Computer Engineering from the University of British Columbia, Vancouver, BC, Canada. Her research interests include computer networking, multimedia communications, cybersecurity and privacy, eHealth, Internet of Things, blockchain, cloud and edge computing. She has published over 80 refereed papers in these areas. She has served as a General Chair/Program Chair/ Technical Program Committee Member for numerous IEEE conferences and workshops, and as a Reviewer for a wide range of high impact research journals and ACM/IEEE flagship conferences.

Yul Chu is a Professor in the Department of Electrical Engineering at University of Texas Rio Grande Valley, USA. He received his Ph.D. in Electrical and Computer Engineering from University of British Columbia, Canada in 2001 and MS in Electrical engineering from Washington State University in 1995. His current research interests include high performance computing, parallel processing, cluster and highavailable architectures, low-power embedded systems, computer networking, digital system design, etc. He has published over 60 refereed papers in these areas.

Li-e Wang is an Associate professor and a doctoral candidate in the College of Computer Science and Information Engineering at Guangxi Normal University, China. She received her Master degrees in Software Engineering from Hunan University in 2007, China. Her research interests mainly include data privacy, computer networking, Healthcare, and distributed system security. She has published over 20 refereed papers in these areas. She has served as a reviewer for several high impact research journals and ACM/IEEE flagship conferences.

Feng Yu is an Associate professor in the College of Computer Science and Information Engineering at Guangxi Normal University, China. Her research interests include trustworthy and controllable network, data security, distributed system security, and Cloud computing. She has published over 20 refereed papers in these areas. She has served as a Technical Program Committee member for several IEEE conferences and workshops.

Xianxian Li is a Professor in the College of Computer Science and Information Engineering at Guangxi Normal University, China. His research interests include data security, distributed system security, Internet of things, and software theory. He has published over 60 refereed papers in these areas. He has served as a Program Co-Chair/Technical Program Committee member for several IEEE conferences and workshops.