Contents lists available at ScienceDirect

Digital Communications and Networks

journal homepage: www.keaipublishing.com/dcan

Users' experience matter: Delay sensitivity-aware computation offloading in mobile edge computing

Mingzhi Wang [a], Tao Wu [a,*], Tao Ma [a], Xiaochen Fan [b], Mingxing Ke [a]

[a] National University of Defense Technology, Hefei, 230031, China
[b] Consulting Center for Strategic Assessment, Academy of Military Science, Beijing, 100091, China

ARTICLE INFO

Keywords:
Mobile edge computing
Computation offloading
Delay sensitivity
Centralized offloading

ABSTRACT

As a promising computing paradigm, Mobile Edge Computing (MEC) provides communication and computing capability at the edge of the network to address the concerns of massive computation requirements, constrained battery capacity and limited bandwidth of the Internet of Things (IoT) systems. Most existing works on mobile edge task ignores the delay sensitivities, which may lead to the degraded utility of computation offloading and dissatisfied users. In this paper, we study the delay sensitivity-aware computation offloading by jointly considering both user's tolerance towards delay of task execution and the network status under computation and communication constraints. Specifically, we use a specific multi-user and multi-server MEC system to define the latency sensitivity of task offloading based on the analysis of delay distribution of task categories. Then, we propose a scoring mechanism to evaluate the sensitivity-dependent utility of task execution and devise a Centralized Iterative Redirection Offloading (CIRO) algorithm to collect all information in the MEC system. By starting with an initial offloading strategy, the CIRO algorithm enables IoT devices to cooperate and iteratively redirect task offloading decisions to optimize the offloading strategy until it converges. Extensive simulation results show that our method can significantly improve the utility of computation offloading in MEC systems and has lower time complexity than existing algorithms.

1. Introduction

With the rapid development of smart devices and communication technologies, IoT devices have been deeply integrated into our daily lives [1–4] to motivate a wide variety of applications. Many resource-consuming applications, such as multi-perception fusion and virtual/augmented reality are more complex than ever, placing demanding requirements for computation and communication resources [5]. Due to their constrained resources, IoT devices usually suffer from a lack of computational power [6] when faced with complex computational tasks, or even rapidly deplete their battery power [7–9]. Although *Cloud Computing* techniques can provide sufficient resources to solve computation-demanding tasks [10], the noticeable delay caused by long transmission is a common frustration for users.

Recently, Mobile Edge Computing (MEC) has emerged as a promising computing paradigm [11,12] that deploys micro servers at the edge of networks in proximity to IoT devices. With a short transmission distance and sufficient bandwidth, MEC can provide computation services and support task execution for IoT devices, thereby enhancing users' experience and QoS [13–15]. A typical computing scenario for MEC systems is shown in Fig. 1, where IoT devices can offload computation tasks to nearby edge servers via communication channels provided by base stations in various urban sites. Based on this classic scenario of edge computing, many existing works have focused on devising various computation offloading strategies with the goal of optimizing offloading decisions and minimizing the average task execution delay [16–18]. However, most of these schemes do not take into account the delay sensitivities of tasks in different categories, leading to the degraded utility of computation offloading to meet the needs of users [19,20].

Generally, IoT devices can generate different types of computation tasks. For instance, in Ref. [21], authors analyzed a cellular traffic dataset collected by a major network operator in China, where requested applications are classified into 19 categories as reflected in Table 1. People have different tolerances for various types of task execution delays, for example, a short delay in an online shopping application is rarely a concern for cell phone users, but the same delay in autonomous driving

* Corresponding author.
E-mail addresses: wang_mzhi@nudt.edu.cn (M. Wang), terence.taowu@gmail.com (T. Wu), chenbbm@sina.com (T. Ma), fanxiaochen33@gmail.com (X. Fan), kemingxing20@nudt.edu.cn (M. Ke).

https://doi.org/10.1016/j.dcan.2022.08.005
Received 22 June 2021; Received in revised form 7 July 2022; Accepted 6 August 2022
Available online 23 August 2022

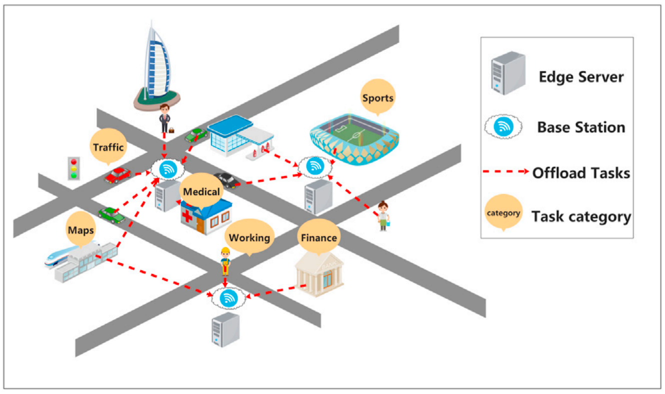


Fig. 1. A typical computing scenario for MEC systems.

can lead to a crash or even endanger human life. It is worth noting that latency sensitivity is different for different types of computing tasks and can be assessed qualitatively based on user surveys. Thus, such differences can be categorized by using three labels (*i.e.*, *sensitive, moderate, insensitive*) to mark delay sensitivities of different application categories.

To illustrate the influence of various sensitivities on the utility of computation offloading, a specific example is shown in Fig. 2. There are a game task and a news task which need to be offloaded to edge servers with two potential strategies. When conducting offloading strategy 1, the game task is offloaded to edge server 1, and the news task will be processed by edge server 2. The execution delays for the game task and the news task are 3 ms and 15 ms, respectively. When conducting offloading strategy 2, the execution delays of both the game task and the news task are 7 ms. Apparently, the average delay of offloading strategy 2 (7 ms) is lower than that of offloading strategy 1 (9 ms). Nevertheless, strategy 2 is not a desirable solution when considering the delay sensitivities of the above tasks since the game task is more sensitive to delay and vice versa for the news task. To conclude, users' experience can be significantly affected by the same delay in executing different application tasks, and when the user experiences longer response times for latency-sensitive tasks, it can cause user dissatisfaction.

Therefore, in this paper, we take delay sensitivity into full consideration to study computation offloading in mobile edge computing system. We aim to optimize the computation offloading strategy with the goal of maximizing the task execution utility. However, there are three critical challenges in finding the best offloading strategy as follows.

- First, delay sensitivity is an abstract concept that is difficult to quantify, so it becomes challenging to evaluate the utility of task execution in terms of conceptual delay sensitivity.
- Second, in MEC systems, heterogeneous edge servers have different computation capacities and communication bandwidth. On this basis, the channel quality and task queues on each edge server change dynamically when IoT devices offload tasks, and it is a challenge to adaptively adjust the offload policy in a dynamic MEC environment.
- Third, when the number of edge servers and tasks increases, the solution space for offloading strategies would become huge, so it is difficult to find an optimal offload policy with the highest utility.

To address the above challenges, first the delay is modeled when considering the following three factors: task uploading, task queuing and task computation. Then, the delay distribution in various task categories is analyzed and their delay sensitivities are defined. Next, the utility function is constructed to quantify the utility as a score so as to evaluate the effect of task execution according to the delay sensitivity of different task categories. To this end, a Centralized Iterative Redirection Offloading (CIRO) algorithm is proposed, which can collect all the information in the MEC system for computation task offloading. Starting from an initial offloading strategy, the CIRO enables IoT devices to cooperate and iteratively redirect task offloading decisions. The contributions of this paper can be summarized as follows.

- We are the first to evaluate the utility of task execution and proceed from the user experience when considering the task execution delay and delay sensitivity for better user satisfaction and user experience.
- We explore a more comprehensive delay composition and more complex dynamic MEC systems, and propose the CIRO algorithm for computational offloading through the collaboration of IoT devices to efficiently and quickly find the offloading strategy with the highest utility in the face of large MEC systems.
- We conduct extensive simulation experiments to verify the effectiveness of the CIRO algorithm and compare it with the existing computation offloading algorithms. Experimental results show that the CIRO algorithm can converge faster and outperform other baseline algorithms for various settings.

The rest of the paper is organized as follows. Section 2 summarizes related works in computation offloading. Section 3 presents the system model and problem formulation. Section 4 introduces the design of our offloading strategy and the CIRO algorithm. Section 5 presents the simulation experiments and the comparison results of the CIRO algorithm with existing algorithms. Section 6 concludes this paper.

2. Related work

Computation offloading is considered as one of the core technologies

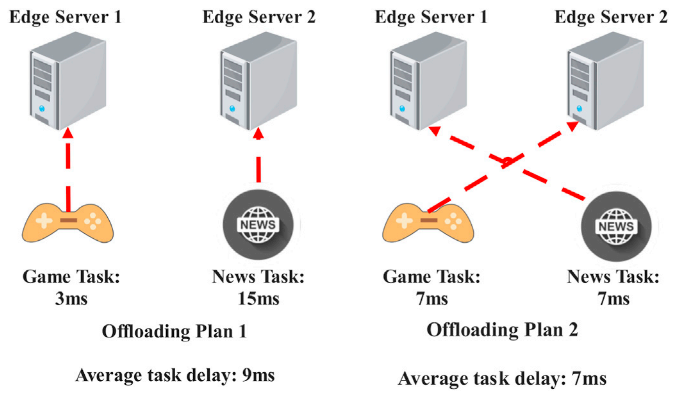


Fig. 2. Two offloading strategies.

Table 1
App classification and delay sensitivities.

No.	Category	Delay sensitivity	No.	Category	Delay sensitivity	No.	Category	Delay sensitivity
1	Games	sensitive	8	Travel	moderate	14	Music	not sensitive
2	Video	moderate	9	Life service	sensitive	15	Maps	sensitive
3	News	not sensitive	10	Education	moderate	16	Reading	not sensitive
4	Social	not sensitive	11	Medical	sensitive	17	Fashion	moderate
5	Shopping	not sensitive	12	Commute	not sensitive	18	Working	sensitive
6	Finance	sensitive	13	Vehicle	sensitive	19	Babycare	sensitive
7	Estate	not sensitive						

in MEC and has attracted significant research attention in recent years. The computation offloading can be divided into three categories according to the optimization goal: (1) minimizing the delay [22,23]; (2) reducing the energy consumption [24–27]; and (3) balancing the delay and energy consumption [28]. In this article, we mainly study computation offloading to reduce the delay.

Existing works on computation offloading can be broadly classified into two categories: centralized and distributed. For centralized computation offloading [29,30], the MEC controller collects all information, including the quality of wireless channels, the queuing status and computing capability of edge servers, and the information of tasks, *etc.* Based on this information, the controller gives offloading decisions for each task. Cooperation among multiple Mobile Devices (MD) is presented in Ref. [29] to realize the task scheduling by constructing a potential game with the aim of minimizing the overhead of each MD. In Ref. [30], Chen et al. propose a task offloading policy for mobile edge computing in software-defined ultra-dense networks. They formulate the task offloading problem as an NP-hard nonlinear mixed-integer programming problem. Then, they transform this optimization problem into a task placement sub-problem and resource allocation sub-problem.

Another thrust of works targets distributed resource allocation for multi-user MEC systems [31,32]. In Ref. [31], they formulated the problem as a distribution overhead minimization problem by targeting multi-user and multi-server scenarios for small cell networks integrated with MEC to minimize the overhead of users. In Ref. [33], authors have studied the impact of inter-user task dependency on the task offloading decisions and resource allocation in a two-user MEC network. They propose an efficient algorithm to optimize resource allocation and task offloading decisions. In Ref. [34], the author provides a fine-grained offloading strategy in mobile edge computing for IoT systems. They propose a novel offloading scheme to reduce the overall completion time of the IoT applications by jointly considering the dependency among subtasks and the contention among multiple users.

In [34,35], the authors use game theory to find the optimal unloading strategy, model computational unloading as a noncooperative game process, and design algorithms so that the game reaches a Nash equilibrium through a finite number of iterations. In a non-cooperative game, IoT devices compete for computing and communication resources with the goal of maximizing their own utility. The difference is that we believe that IoT devices can use various resources efficiently by cooperating, which can avoid the bad consequences of vicious competition caused by non-cooperative games.

3. System model

This section introduces the system model. We first describe the multi-user and multi-server MEC system, and then present communication model, computation model, and task execution utility model in detail. For clear presentation, Table 2 summarizes the notations used in the following formulation.

3.1. Multi-user and multi-server MEC system

This paper considers the MEC system with multiple users and multiple edge servers. An illustration of the MEC system is shown in Fig. 3. It is assumed that there is a set of $N = \{1, 2, ..j, ..n\}$ IoT devices. Edge servers distributed at the edge of the network can be defined by $S = \{1, 2, ..k, ..s\}$. In the MEC system, edge servers are located at base stations and IoT devices can offload tasks through radio channels provided by base stations. $M = \{1, 2, ..i, ..m\}$ are tasks generated by all IoT devices. As analyzed above, tasks can be divided into different types based on delay sensitivity. Therefore, we use $\Upsilon = (\upsilon_1, \upsilon_2.., \upsilon_h, ..\upsilon_l)$ to represent the categories of task set M, υ_h is the $h-th$ task category.

T_i represents the $i-th$ task, which can be described by a five-tuple $(m_i, c_i, a_i, w_i, u_i)$. Here m_i denotes the size of input data for computation (e.g., the program codes and inputting parameters), c_i denotes the

Table 2
Notations used in the paper.

Notation	Definition
N	The set of IoT devices
S	The set of edge servers
M	All tasks in the MEC system
T_i	$j-th$ task in the MEC system
m_i	The computation input data size of T_i
c_i	The required CPU cycles of T_i
a_i	The offloading decision of T_i
w_i	The IoT device that generated T_i
u_i	The category that T_i belongs to
A	The offloading strategy of all tasks
B_k	The bandwidth of BS connected to edge server k
q_j	The transmission power of IoT device j
$g_{j,k}$	Channel gain
f_j^l	The computation capability of IoT device j
f_k^e	The computation capability of edge server k
t_i	Total delay of T_i
Φ	The delays of all tasks
υ_h	The $h-th$ task category
γ_h	Delay distribution of $h-th$ task category
μ_h	The mean of γ_h
σ_h^2	The variance of γ_h
d_i	The execution score of T_i

required total CPU cycles to accomplish the task T_i. The value of a_i denotes where task T_i is executed, $a_i = 0$ means the task T_i is executed locally in the IoT device. $a_i = k$ means T_i is offloaded to the edge server k for execution. w_i indicates the IoT device that generated the task T_i. u_i denotes the category which task T_i belongs to. Offloading strategy includes offloading decisions of all tasks and can be denoted by $A = (a_1, a_2, .., a_i, .., a_m)$.

3.2. Communication model

The communication model for wireless access is presented as

$$r_{i,k} = B_k log_2\left(1 + \frac{q_i g_{i,k}}{\varpi + \sum_{i' \in M: a_{i'} = a_i} q_{i'} g_{i',k}}\right) \tag{1}$$

B_k represents bandwidth of the channel connected to edge server k. $r_{i,k}$ denotes the upload data rate from device i to edge server k. Here q_i is the transmission power of device i, $g_{i,k}$ means channel gain for the link between device i and edge server k. ϖ is background noise power. $\sum_{i' \in M: a_i' = a_i} q_{i'} g_{i',k}$ denotes the wireless interference suffered from other IoT devices which offloads tasks to the same edge servers. It can be seen that $r_{i,k}$ is determined by offloading strategy A. If an edge server serves more IoT devices, the upload rate from the IoT device to the edge server will be lower, leading to an increase in transmission delay.

3.3. Computation model

For each task, there are two kinds of execution methods. In the first method, tasks are executed on the IoT devices that generate them called *Local Computing*. The second method is to offload tasks to edge servers for execution called *Edge Computing*. Two different task execution modes produce various delays.

1) *Local computing*: Task T_i chooses *Local Computing* will only generate computing delay. We know the required total CPU cycles of task T_i, f_j^l represents the computation capability (i.e., CPU cycles per second) of IoT device j that generated task T_i, $w_i = j$. Then we can get the computing delay t_i^l of T_i as the following formula:

$$t_i^l = c_i \Big/ f_j^l \tag{2}$$

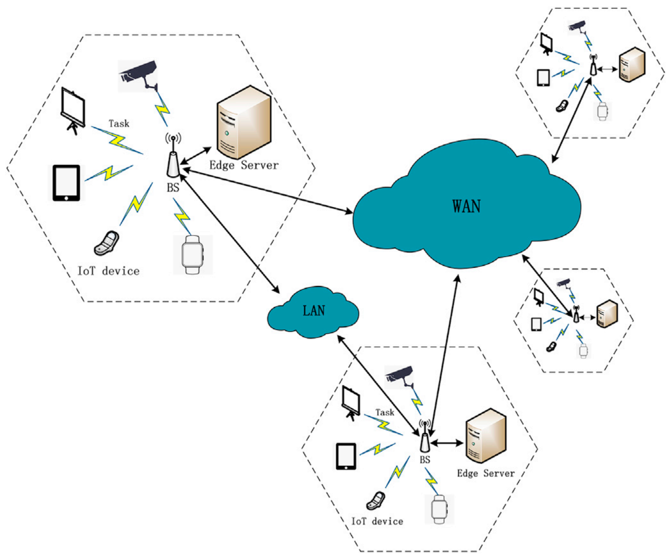


Fig. 3. An illustration of the MEC system.

2) *Edge computing*: Tasks' *Edge Computing* can be divided into three steps. First, the task will be uploaded to an edge server, then the task queued in the edge server for execution, and finally, the processor calculates the task to get results.

In the first step, task T_i is uploaded from the IoT device j to an edge server and causes upload delay $t_i^{e,upload}$. Given the offloading strategy A, we can get the upload data rate $r_{j,k}$ by formula (1). Therefore, the upload delay $t_i^{e,upload}$ of T_i can be calculated as

$$t_i^{e,upload} = m_i / r_{j,k} \tag{3}$$

In the second step, when T_i arrives on the edge server k, there may be some tasks waiting to be executed. Without the execution order of tasks on edge server k, the waiting delay $t_i^{e,wait}$ of T_i is difficult to calculate. Therefore, we take a worst-case scenario and assume that T_i is executed last in the edge server to estimate the waiting delay as follows

$$t_i^{e,wait} = \sum_{i' \in M : a_{i'} = a_i} c_{i'} \Big/ f_k^e \tag{4}$$

In this way, we can get the longest waiting delay of task T_i on each edge server, based on which we can select the appropriate edge server for T_i. Here, f_k^e is the computation capability of the edge server k, $\sum_{i' \in M : a_{i'} = a_i} c_{i'}$ is the sum of CPU cycles of tasks queued before T_i in the edge server.

In the third step, the computing delay $t_i^{e,compute}$ on the edge server can be calculated by

$$t_i^{e,compute} = c_i / f_i^e \tag{5}$$

The total delay t_i^e of task T_i in *Edge Computing* is composed of uploading delay, waiting delay and computing delay as

$$t_i^e = t_i^{e,upload} + t_i^{e,wait} + t_i^{e,compute} \tag{6}$$

t_i represents the total delay of T_i, when the task chooses *Local Computing*, $t_i = t_i^l$, when the task chooses *Edge Computing*, $t_i = t_i^e$. Given the offloading strategy, the channel interference and task queue on each edge server can be determined. So we can calculate the delays $\Phi = (t_1, t_2, .., t_i, .., t_m)$ of all tasks through the above delay calculation method.

3.4. Task execution utility model

As mentioned above, our goal is to find the offloading strategy with the highest utility, so we first propose a method to evaluate the utility of the task execution in terms of delay and delay sensitivity. The delay distribution model of a task category can reflect its delay sensitivity and we consider that the execution delays of the same task category all obey Gaussian distribution. The Gaussian distribution model for different task categories can be expressed as $\Gamma = [\gamma_1, \gamma_2.., \gamma_h.., \gamma_l]$, γ_h denotes Gaussian distribution for task category υ_h, μ_h is the mean of γ_h, and σ_h^2 is the variance. The task category with the smaller mean and variance of the delay distribution has higher delay sensitivity.

We compare the task's execution delay with the delay distribution to evaluate its execution utility. Next, we take task T_i as an example to introduce the evaluation process. It is assumed that task T_i belongs to type υ_h and γ_h is its Gaussian distribution model. The probability density function of γ_h can be expressed by

$$f_h(x) = \frac{1}{\sqrt{2\pi}\sigma_h} \exp\left(-\frac{(x-\mu_h)^2}{2\sigma_h^2}\right) \tag{7}$$

We assume that the mean μ_h is 500 ms and σ_h is 100 ms, and then we can draw the graph of γ_h as shown in Fig. 4. After calculating the execution delay t_i of task T_i, we compare t_i with γ_h to analyze the execution utility of T_i. The shaded area in Fig. 4 represents the probability $p_h(t_i)$ that the execution utility of T_i is better than other tasks in type υ_h, $p_h(t_i) = P\{X > t_i\}$ can be calculated by

$$p_h(t_i) = \int_{t_i}^{\infty} \frac{1}{\sqrt{2\pi}\sigma_h} \exp\left(-\frac{(x-\mu_h)^2}{2\sigma_h^2}\right) \tag{8}$$

Based on this comparison, we use a scoring mechanism to quantify the execution utility of the task. d_i denotes the score of task T_i's execution utility, which can be represented by

$$d_i = p_h(t_i) \times 100 \tag{9}$$

This scoring mechanism evaluates the execution utility by considering both execution delay and delay sensitivity of tasks. The utility of the offloading strategy can be reflected by the average of all tasks' scores, which is called *AverageScore* and can calculate by

$$AverageScore = \left(\sum_{i' \in M} d_i\right) \Big/ m \tag{10}$$

Fig. 5 shows how the scores of the execution utility in different task categories vary with delay. It can be seen that the scores of delay-sensitive tasks decrease rapidly with the growth of delay. To get a higher *AverageScore*, the offloading strategy is adjusted so that latency-sensitive tasks are executed as soon as possible, which is exactly what users need.

4. Problem formulation

In the MEC system, different offloading strategies will bring completely different results. The offloading strategy $A = (a_1, a_2, .., a_i, .., a_m)$ determines the execution position of all tasks, and the execution position of all tasks will affect *AverageScore*. According to A, we can calculate tasks' execution delay $\Phi = (t_1, t_2, .., t_i, .., t_m)$ as in the 3.3 *Computing Model*. Based on Φ, *AverageScore* can be calculated by formulas (8), (9), (10), which reflects the utilities of the offloading strategy.

Our goal is to find the optimal offloading strategy with the highest *AverageScore*, the task offloading problem can be formulated as

$$\textbf{\textit{Opt}}. \quad \max AverageScore = \\ = 100\left(\sum_{t_i \in \Phi}\int_{t_i}^{\infty}\frac{1}{\sqrt{2\pi}\,\sigma_h}\exp\left(-\frac{(x-\mu_h)^2}{2\sigma_h^2}\right)\right)\Big/ m \tag{11}$$

5. Task offloading strategy

This section proposes an efficient centralized algorithm to solve the optimization problem defined above. Unlike the distributed method, the centralized approach can obtain information about the MEC system and offload tasks based on a global perspective.

5.1. The CIRO algorithm

The CIRO algorithm chooses an edge server as the central station, which collects global information, based on which, the central station server runs the CIRO algorithm to get the offloading strategy with the highest *AverageScore* and informs IoT devices of the task's offloading decision by broadcasting. Then, IoT devices offload tasks according to the central station's commands. The core idea of the CIRO algorithm can be briefly expressed as follows: After collecting global information, the CIRO algorithm generates an initial offloading strategy denoted by A_0, then iteratively adjusts the task's offloading decision to improve the *AverageScore* of the offloading strategy. A_l represents the offloading strategy after the $l-th$ iteration. When an iteration is adjusted to no longer increase the AverageScore, the CIRO algorithm converges and obtains the final offloading policy. The main process of the CIRO algorithm can be divided into three steps as follows.

(1) *Global information collection*: At the beginning of the algorithm, the IoT devices send five-tuple information of tasks to the central station server. The edge server will also send the server's information, including the server's computing capability, task queue status, base station's bandwidth, and other information to the central station. The data amount of this information is small that will not occupy too many communication resources.
(2) *Initial offloading strategy generation*: After collecting the global information, the CIRO algorithm initializes the offloading strategy A_0 in a random way, that is, randomly generating the offloading decision for each task (**line 1**). Starting from A_0, the CIOR algorithm then iteratively adjusts the offloading strategy. When the offloading strategy does not change after the iteration, the algorithm is considered to converge and ends (**line 2**).

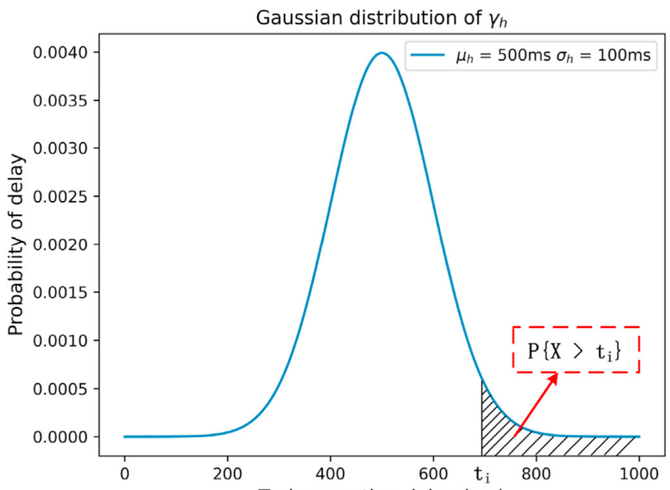


Fig. 4. Delay sensitivity analysis of task T_i.

(3) *Iterative optimization*: In the iterative process, the adjustment method of the offloading strategy is to redirect the offloading decisions of tasks. Redirection of task refers to changing the task's current offloading decision to find a better offloading decision that makes offloading strategy's *AverageScore* higher. In order to reduce the complexity of the redirection, we only redirect one task at a time, and the offloading decision of other tasks remains unchanged during this time.

Algorithm 1: Centralized iterative redirection offloading (CIRO) algorithm.

```
Input: Task Information, Edge Server Information
Output: The offloading strategy with the highest AverageScore
1  Initialize $A_0$, $a_i$ is a random value in $\{0\} \cup S$
2  while $A_l \neq A_{l-1}$ do
3      for all $i \in M$ do
4          for all $j \in M$ and $j \neq i$ do
5              Attempt to swap the offloading decisions of the two tasks, swap the values of $a_i$ and $a_j$ in $A_l$ to get $A_{l,i}^1$
6              Calculate AverageScore of $A_{l,i}^1$
7          end
8          Get $A_{l,i}^1$ with highest AverageScore
9          for all $k \in S$ do
10             Attempt to redirect task $T_i$ to server $S_k$, let $a_i = k$ in $A_l$ get $A_{l,i}^2$.
11             Calculate AverageScore of $A_{l,t}^2$
12         end
13         Get $A_{l,i}^2$ with highest AverageScore
14         Attempt to redirect task $T_i$ from the edge server to local, $a_i = 0$ or from local to an edge server to get $A_{l,i}^3$.
15     end
16     Choose the offloading strategy with the highest AverageScore among $A_{l,i}^1$, $A_{l,i}^2$, $A_{l,i}^3$ as $A_{l,i}^*$
17     if AverageScore of $A_{l,i}^*$ >AverageScore of $A_l$ then
18         Implement that redirection and update $A_l = A_{l,i}^*$
19     end
20     $l = l + 1$
21 end
22 Get the final offloading strategy and notify IoT devices to offload according to the strategy through broadcast
```

In a round of iteration, we will attempt to redirect the task to each possible offloading decision called redirection attempts, to find the best redirection, and the redirection attempts for each task can be divided into three types. The attempt will result in changes to the offloading strategy. We use $A_{l,i}^j$ to represent the offloading strategy after the $j-th$ type redirection attempt of T_i in the $l-th$ iteration. Next, take task T_i as an example to introduce three types of redirection processes.

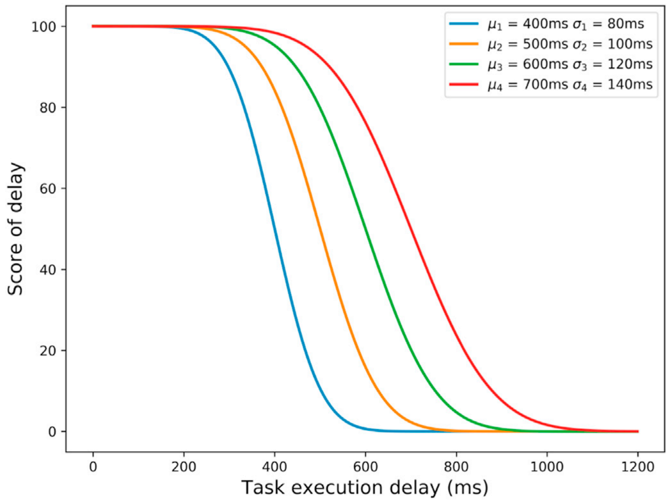


Fig. 5. Score of delay in different task categories.

- As shown in Fig. 6(a), the first type of redirection is to swap T_i's offloading decision with other tasks (**lines 4–8**). After the swapping, *AverageScore* will increase or decrease. We let T_i attempt to swap the decision with all tasks to find the offloading strategy $A^1_{l,i}$ with the highest *AverageScore* after first type redirection attempt.
- The second type of redirection (Fig. 6(b)) is simply to change the offloading decision of task T_i from the current edge server to other edge servers to get the offloading strategy $A^2_{l,i}$ (**lines 9–13**). At last, we will get the offloading strategy $A^2_{l,i}$ with the highest *AverageScore*.
- As shown in Fig. 6(c), the third kind of redirection is to redirect the offloading decision of task T_i from the current edge server to the local or from the local to an edge server (**line 14**).

After completing the three types of redirection attempts, we get the offloading strategy ($A^1_{l,i}$,$A^2_{l,i}$,$A^3_{l,i}$) with the highest *AverageScore* in each kind of redirection attempt. Then we choose the best offloading strategy in ($A^1_{l,i}$,$A^2_{l,i}$,$A^3_{l,i}$), which can be denoted by $A^*_{l,i}$ (**line 16**). If the *AverageScore* of $A^*_{l,i}$ exceeds the *AverageScore* of offloading strategy A_l, then we update the offloading strategy by $A_l = A^*_{l,i}$ (**line 17–18**), otherwise, we will not update A_l. Because the purpose of task's redirection is to increase the *AverageScore* of the offloading strategy, we will not implement the redirection without the increase of *AverageScore*.

When all tasks are redirected, this round of iteration ends. When the *AverageScore* cannot be improved by task redirection, the offloading strategy in the iteration will not change, and the CIRO algorithm converges. It is worth noting here that redirection is only part of the calculation in the algorithm, rather than actually offloads the task.

5.2. Convergence analysis of CIRO

Next, we prove that the CIRO algorithm can converge in a finite number of iterations. We use C to represent the number of iterations of the CIRO algorithm. C may be a finite integer or infinite. Because finding the optimal offloading strategy in the MEC system is a combination problem, therefore there is the optimal offloading strategy $A^* = (a^*_1, a^*_2, ..., a^*_i, ..a^*_n)$ with the highest *AverageScore*. The CIRO algorithm continuously improves *AverageScore* of offloading strategy through iterations, and Δ_t represents the increment of *AverageScore* after the t-th iteration. Let $\Delta_{\min}$ be the smallest increment of all iterations in the CIRO algorithm, $\Delta_{\min} = \min \Delta_d$, $0<d \le C$. Because the *AverageScore* of each offloading strategy is a certain value, $\Delta_{\min}$ cannot be infinitely small. So there must be a positive real number ϵ that ϵ is smaller than $\Delta_{\min}$, $\epsilon<\Delta_{\min}$.

We use $AverageScore^0$ to represent the *AverageScore* of the initial offloading strategy A_0. *AverageScore** represents the *AverageScore* of the optimal offloading strategy A^*. Based on $AverageScore^*<100$ and $\epsilon<\Delta_{\min}$, we know that

$$C < \frac{(AverageScore^* - AverageScore^0)}{\Delta_{min}} < \frac{100}{\epsilon} \tag{12}$$

Because ϵ is a positive real number that is not infinitesimal, $100/\epsilon$ is a finite real number. Through formula (12), we can deduce that the CIRO algorithm can converge through a limited number of iterations.

5.3. Complexity analysis of CIRO

After the convergence analysis, we use the calculation of *AverageScore* as the basic operation to analyze the complexity of the CIRO algorithm in an iteration. It is assumed that there are n tasks that need to be offloaded to the k edge servers or local, and in each iteration, we perform three redirect attempts for all n tasks.

In the first redirection attempt, each task has to swap offloading decisions with other $n-1$ tasks. In the second redirection attempt, each task attempts to be redirected to other servers for computing, and $k-1$ basic operations are implemented. In the third redirection attempt, the task is attempted to be redirected to local or from local to an edge server, and only implements 1 basic operation. So each task has to do $n+k-1$ basic operations. In an iteration, n tasks' redirection require $n(n+k-1)$ basic operations, so the complexity of the CIRO algorithm in each iteration is $O(n(n+k))$. Besides that, a few tasks change the offloading decision in the redirection process. When calculating *AverageScore*, only the scores of the affected tasks need to be recalculated, which significantly reduces the amount of computation.

6. Experiment and evaluation

In this section, we conduct experimental simulations to evaluate the performance of the proposed CIRO algorithm. Under three different scenarios, we calculate *AverageScore* of the offloading strategy to verify the effectiveness of our method. In particular, the Distributed Earliest Finish time Offload (DEFO) and the Random algorithm are used to compare with the CIRO algorithm, which randomly offloads tasks to edge servers for execution.

6.1. Experimental parameter setting

The experiment is set in a multi-user and multi-server scenario. The parameter settings in the experiments are shown in Table 3. We set the channel bandwidth of each base station as 20 *Mhz* ± 20% and the computing capability of each edge server as 10 *Ghz* ± 20%. Considering the QoS of tasks, the number of tasks queued on the edge server is ranged from 0 to 10 randomly. We set the computing capability of IoT devices to 1 *Ghz* ± 20%, and the transmission power of each device as 100*mWatts* ± 20%. Distance between IoT devices and edge servers ranges from 20 m to 50 m at random. For tasks generated by IoT devices, the task's data volume is 1000 KB ± 50%, and the number of CPU cycles required for the task is between [1000,5000] Megacycles. The background noise is assumed as −100 dBm and the pass loss factor is 4. We divide tasks into 10 species and the mean of each task specie's delay distribution is 500 ms ± 30%, and the standard deviation is 120 ms ± 30%.

6.2. AverageScore of different algorithms

As mentioned above, our purpose is to obtain an offloading strategy with the highest *AverageScore*. Hence, we first use the *AverageScore* as an evaluation index to verify the effect of algorithms. Here, three experiments are designed to study the impacts of the different numbers of edge servers, IoT devices, and tasks on *AverageScore*.

In the first experiment, we aim to study the impact of different numbers of edge servers on *AverageScore*. We set the number of IoT devices to 15, the number of tasks to 50, then gradually increase the number of edge servers from 1 to 20. The *AverageScore* of offloading strategy generated by each algorithm is shown in Fig. 7(a). When there are only a few edge servers, many tasks will be queued on them and cause task waiting delay. Therefore, the *AverageScore* of offloading strategy will decrease. As the number of edge servers increases, computing resources become more sufficient, and the*AverageScore* gradually becomes higher. It can be seen from Fig. 7(a) that the *AverageScore* of three algorithms increases simultaneously with a larger number of edge servers. The *AverageScore* of the CIRO algorithm are always higher than those of DEFO algorithm and Random algorithm. The above results show that the proposed CIRO algorithm can achieve robust performance under different scales of edge networks.

The second experiment studies the relationship between the *AverageScore* and the number of IoT devices. We set the number of edge servers at 10, and the number of IoT devices gradually increased from 1 to 30. Since the number of tasks is related to the number of IoT devices, we set the number of tasks as three times the number of IoT devices. With experimental results in Fig. 7(b), we find that as the number of IoT devices increases, the competition for edge servers becomes more intense, resulting in more interferences in communication channels and longer task queues on each edge server. However, the drop rate of *AverageScore* by the CIRO algorithm is significantly slower than the two baseline algorithms. In addition, the CRIO can still achieve a relatively satisfactory *AverageScore* with a large number of IoT devices.

In the third experiment, we further explore the impact of tasks' number on *AverageScore*. For this simulation, the number of tasks gradually increases from 1 to 50 in an MEC system with 10 edge servers and 15 IoT devices. The experimental results shown in Fig. 7(c) are similar to the second experiment. As the number of tasks increases, the *AverageScore* becomes to decline for three algorithms, however, the CIRO algorithm significantly outperforms two baseline methods with a steady *AverageScore*.

Table 3
Experimental parameters.

Parameters	Value
Base station bandwidth	20 *Mhz* ± 20%
Computing capability of edge servers	10 *Ghz* ± 20%
Number of waiting tasks	[0,10]
Computing capability of IoT devices	1 *Ghz* ± 20%
Transmission power of IoT devices	100 *mWatts* ± 20%
Distance from IoT device to each server	[20,50]m
Task data volume	1000 KB ± 50%
CPU cycles number required for the task	[1000,5000]Megcycles
Background nosie	−100 dBm
pass loss factor	4
Mean of delay distribution	600 ms ± 30%
Standard deviation of delay distribution	120 ms ± 30%

6.3. *Convergence of different algorithms*

In this part, we verify the convergence of the CIRO algorithm and DEFO algorithm. The convergence of the algorithm is reflected in the number of iterations, and an algorithm that achieves convergence with fewer iterations can have lower time complexity in obtaining optimized offloading strategies.

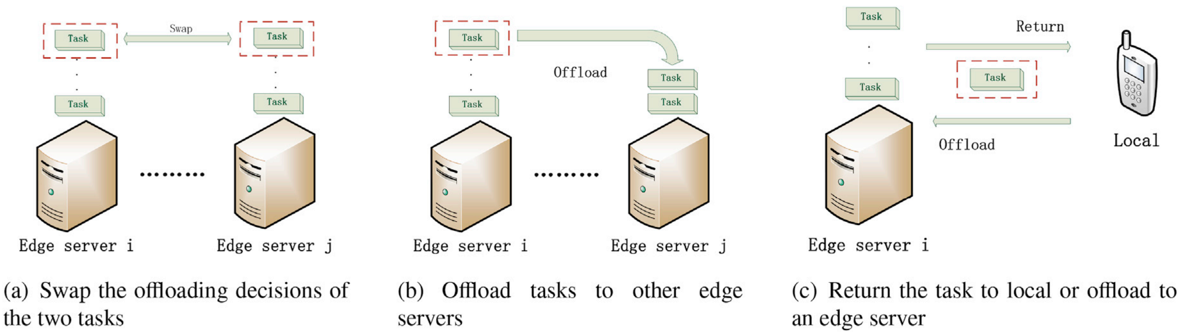
In order to verify the convergence of the two algorithms in different scenarios, three experiments were conducted in different cases. The results of the first experiment are shown in Fig. 8(a). With a larger number of edge servers, the solution space of the offloading strategy also increases, leading to more iterations of algorithms. The number of iterations of the DEFO algorithm increases sharply as the number of edge servers increases. The difference is that the number of iterations in the CIRO algorithm is only a few, which indicates that the CIRO algorithm can efficiently cope with a larger number of edge servers.

The second experiment and the third experiment investigate the changes in the number of iterations for these two algorithms when the number of IoT devices and tasks increase, respectively. As shown in Fig. 8(b) and (c), as the number of IoT devices and tasks increases, the number of iterations in the two algorithms increases accordingly. Meanwhile, judging from the curves of the two algorithms in figures, the CIRO algorithm performs better than the DEFO algorithm, and can reach convergence with fewer iterations.

AverageScore reflects the utilities of the offloading strategy generated by the algorithm. Here, we want to analyze the execution utilities of each task. Therefore, an experiment is designed to observe the score of each task in the two algorithms with changes in the number of edge servers. In the experiment, we set the number of IoT devices to 15 and the number of tasks to 50. We analyze the scores of 50 tasks in the experiment after the execution of the CIRO and DEFO algorithms. Fig. 9 shows the scores of 50 tasks under different numbers of servers. It shows that tasks' scores in the CIRO algorithm are generally higher than that in the DEFO algorithm. As the number of edge servers increases, the proportion of high-scoring tasks in the 50 tasks increases in the CIRO algorithm. The CIRO algorithm improves the score of a single task while offloading with the goal of maximizing *AverageScore*. The DEFO algorithm uses non-cooperative game theory to ensure the utility of a single IoT device, but the competition between IoT devices leads to a drop in the scores of tasks.

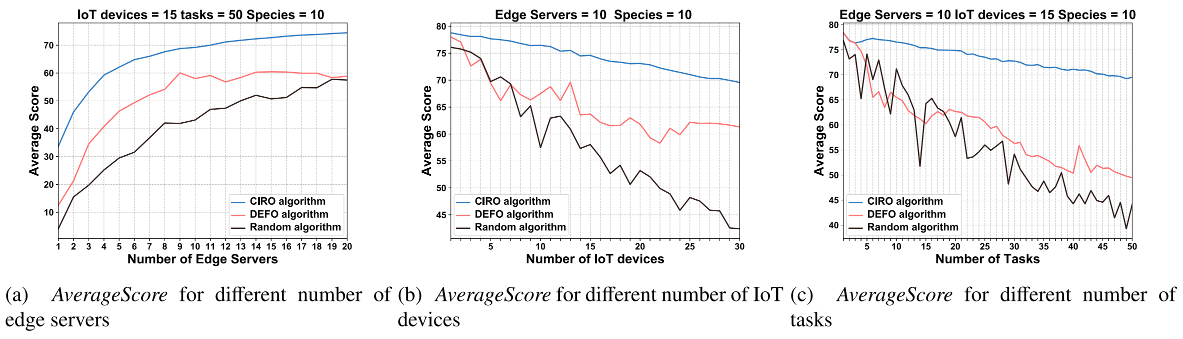
7. Conclusion

In this paper, the problem of computation offloading in the MEC for IoT systems is studied, the user's tolerance towards the task's execution delay is considered and the execution utilities of tasks based on tasks' delay sensitivities are evaluated. To incorporate different IoT devices for efficient computation offloading, the CIRO algorithm is proposed to collect global information and find the optimal offloading strategy through IoT devices' collaboration. Extensive simulations are conducted and the results show that the CIRO algorithm can significantly improve



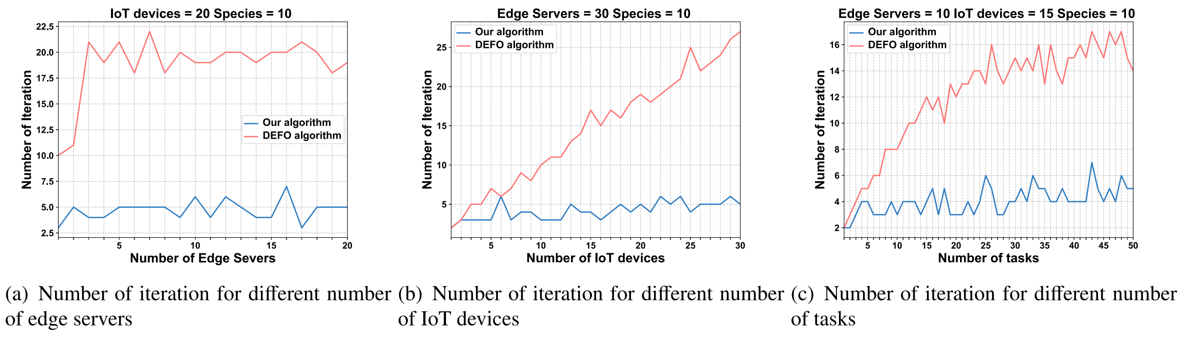
(a) Swap the offloading decisions of the two tasks (b) Offload tasks to other edge servers (c) Return the task to local or offload to an edge server

Fig. 6. Three types of redirect attempts.



(a) *AverageScore* for different number of edge servers (b) *AverageScore* for different number of IoT devices (c) *AverageScore* for different number of tasks

Fig. 7. *AverageScore* of different algorithms.



(a) Number of iteration for different number of edge servers (b) Number of iteration for different number of IoT devices (c) Number of iteration for different number of tasks

Fig. 8. Convergence of different algorithms.

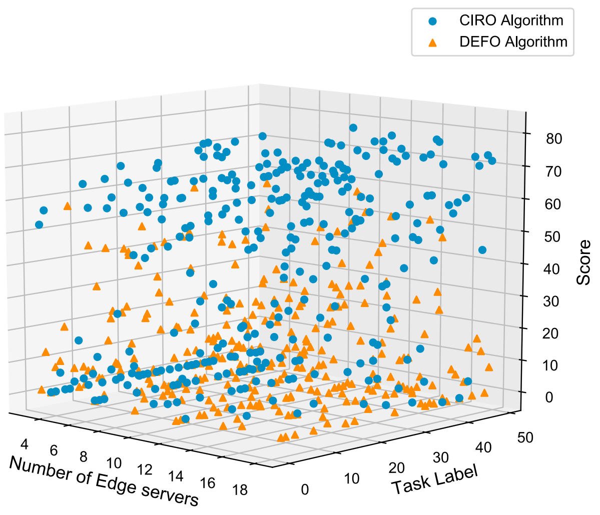


Fig. 9. The scores of tasks in two algorithms.

the utility of the task offloading strategy with lower time complexity, stronger scalability and higher robustness.

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.

Acknowledgments

This research is partially supported by the Hong Kong Scholars Program with No. 2021-101, in part by the National Natural Science Foundation of China under Grant No. 62002377, 62072424, 61772546, 61625205, 61632010, 61751211, 61772488, 61520106007, Key Research Program of Frontier Sciences, CAS, No. QYZDY-SSW- JSC002, NSFC with No. NSF ECCS-1247944, and NSF CNS 1526638, in part by the National key research and development plan No. 2017YFB0801702, 2018YFB1004704.

References

[1] V. Tyagi, A. Kumar, Internet of things and social networks: a survey, in: 2017 International Conference on Computing, Communication and Automation, ICCCA), 2017, pp. 1268–1270, https://doi.org/10.1109/CCAA.2017.8230013.
[2] X. Fan, C. Xiang, C. Chen, P. Yang, L. Gong, X. Song, P. Nanda, X. He, Buildsensys: reusing building sensing data for traffic prediction with cross-domain learning, IEEE Trans. Mobile Comput. 20 (6) (2021) 2154–2171, https://doi.org/10.1109/TMC.2020.2976936.
[3] Y. Yin, Q. Huang, H. Gao, Y. Xu, Personalized apis recommendation with cognitive knowledge mining for industrial systems, IEEE Trans. Ind. Inf. 17 (9) (2021) 6153–6161, https://doi.org/10.1109/TII.2020.3039500.
[4] X. Yang, S. Zhou, M. Cao, An Approach to Alleviate the Sparsity Problem of Hybrid Collaborative Filtering Based Recommendations: the Product-Attribute Perspective from User Reviews, Mobile Networks and Applications.
[5] C. You, K. Huang, H. Chae, Energy efficient mobile cloud computing powered by wireless energy transfer, IEEE J. Sel. Area. Commun. 34 (5) (2016) 1757–1771, https://doi.org/10.1109/JSAC.2016.2545382.
[6] S. Yi, C. Li, Q. Li, A survey of fog computing: concepts, applications and issues, in: Proceedings of the 2015 Workshop on Mobile Big Data, Mobidata '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 37–42, https://doi.org/10.1145/2757384.2757397.
[7] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (iot): a vision, architectural elements, and future directions, including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond, Future Generat. Comput. Syst. 29 (7) (2013) 1645–1660, https://doi.org/10.1016/j.future.2013.01.010.

[8] I. Lee, K. Lee, The internet of things (iot): applications, investments, and challenges for enterprises, Bus. Horiz. 58 (4) (2015) 431–440, https://doi.org/10.1016/j.bushor.2015.03.008.

[9] H. Gao, X. Qin, R.J.D. Barroso, W. Hussain, Y. Xu, Y. Yin, Collaborative learning-based industrial iot api recommendation for software-defined devices: the implicit knowledge discovery perspective, IEEE Trans. Emerg. Top. Comput. Intell. (2020) 1–11, https://doi.org/10.1109/TETCI.2020.3023155.

[10] Y. Huang, H. Xu, H. Gao, X. Ma, W. Hussain, Ssur: an Approach to Optimizing Virtual Machine Allocation Strategy Based on User Requirements for Cloud Data Center.

[11] H.T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches, Wireless Commun. Mobile Comput. 13 (18) (2013) 1587–1611, https://doi.org/10.1002/wcm.1203, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/wcm.1203.

[12] S. Deng, Z. Xiang, P. Zhao, J. Taheri, H. Gao, J. Yin, A.Y. Zomaya, Dynamical resource allocation in edge for trustable internet-of-things systems: a reinforcement learning method, IEEE Trans. Ind. Inf. 16 (9) (2020) 6103–6113, https://doi.org/10.1109/TII.2020.2974875.

[13] X. Lin, Y. Lu, J. Deogun, S. Goddard, Real-time divisible load scheduling for cluster computing, in: 13th IEEE Real Time and Embedded Technology and Applications Symposium, RTAS'07), 2007, pp. 303–314, https://doi.org/10.1109/RTAS.2007.29.

[14] A. Mamat, Y. Lu, J. Deogun, S. Goddard, An efficient algorithm for real-time divisible load scheduling, in: 2010 16th IEEE Real-Time and Embedded Technology and Applications Symposium, 2010, pp. 323–332, https://doi.org/10.1109/RTAS.2010.29.

[15] Y. Yin, Z. Cao, Y. Xu, H. Gao, Z. Mai, Qos prediction for service recommendation with features learning in mobile edge computing environment, IEEE Trans. Cogn. Commun. Netw. 6 (4) (2020) 1136–1145.

[16] Y. Miao, G. Wu, M. Li, A. Ghoneim, M. Al-Rakhami, M.S. Hossain, Intelligent task prediction and computation offloading based on mobile-edge cloud computing, Future Generat. Comput. Syst. 102 (2020) 925–931, https://doi.org/10.1016/j.future.2019.09.035.

[17] Y. Mao, J. Zhang, K.B. Letaief, Dynamic computation offloading for mobile-edge computing with energy harvesting devices, IEEE J. Sel. Area. Commun. 34 (12) (2016) 3590–3605, https://doi.org/10.1109/JSAC.2016.2611964.

[18] Y. Li, H. Ma, L. Wang, S. Mao, G. Wang, Optimized content caching and user association for edge computing in densely deployed heterogeneous networks, IEEE Trans. Mobile Comput. (99) (2020), 1–1.

[19] K. Li, M. Tao, Z. Chen, A computation-communication tradeoff study for mobile edge computing networks, in: 2019 IEEE International Symposium on Information Theory, ISIT), 2019, pp. 2639–2643, https://doi.org/10.1109/ISIT.2019.8849601.

[20] W. Shi, J. Zhang, R. Zhang, K. Hu, An area-based offloading policy for computing offloading in mec-assisted wireless mesh network, in: 2019 IEEE/CIC International Conference on Communications in China, ICCC), 2019, pp. 507–511, https://doi.org/10.1109/ICCChina.2019.8855931.

[21] M. Zeng, T.-H. Lin, M. Chen, H. Yan, J. Huang, J. Wu, Y. Li, Temporal-spatial mobile application usage understanding and popularity prediction for edge caching, IEEE Wireless Commun. 25 (3) (2018) 36–42, https://doi.org/10.1109/MWC.2018.1700330.

[22] C. Yi, J. Cai, Z. Su, A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications, IEEE Trans. Mobile Comput. 19 (1) (2020) 29–43, https://doi.org/10.1109/TMC.2019.2891736.

[23] Y. Nan, W. Li, W. Bao, F.C. Delicato, P.F. Pires, Y. Dou, A.Y. Zomaya, Adaptive energy-aware computation offloading for cloud of things systems, IEEE Access 5 (2017) 23947–23957, https://doi.org/10.1109/ACCESS.2017.2766165.

[24] X. Liu, N. Ansari, Green/Energy-Efficient Design for D2D, American Cancer Society, 2019, pp. 1–24, https://doi.org/10.1002/9781119471509.w5GRef184.

[25] H. Liu, L. Cao, T. Pei, Q. Deng, J. Zhu, A fast algorithm for energy-saving offloading with reliability and latency requirements in multi-access edge computing, IEEE Access 8 (2020) 151–161, https://doi.org/10.1109/ACCESS.2019.2961453.

[26] S. Xia, Z. Yao, Y. Li, S. Mao, Online distributed offloading and computing resource management with energy harvesting for heterogeneous mec-enabled iot, IEEE Trans. Wireless Commun. (2021) p.1, https://doi.org/10.1109/TWC.2021.3076201, 1–1.

[27] Y. Li, C. Liao, Y. Wang, C. Wang, Energy-efficient optimal relay selection in cooperative cellular networks based on double auction, IEEE Trans. Wireless Commun. 14 (8) (2015) 4093–4104.

[28] X. Lan, L. Cai, Q. Chen, Execution latency and energy consumption tradeoff in mobile-edge computing systems, in: 2019 IEEE/CIC International Conference on Communications in China (ICCC), 2019, pp. 123–128, https://doi.org/10.1109/ICCChina.2019.8855969.

[29] L. Tianze, W. Muqing, Z. Min, L. Wenxing, An overhead-optimizing task scheduling strategy for ad-hoc based mobile edge computing, IEEE Access 5 (2017) 5609–5622, https://doi.org/10.1109/ACCESS.2017.2678102.

[30] M. Chen, Y. Hao, Task offloading for mobile edge computing in software defined ultra-dense network, IEEE J. Sel. Area. Commun. 36 (3) (2018) 587–597, https://doi.org/10.1109/JSAC.2018.2815360.

[31] X. Ma, C. Lin, X. Xiang, C. Chen, Game-theoretic analysis of computation offloading for cloudlet-based mobile cloud computing, in: Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 271–278, https://doi.org/10.1145/2811587.2811598.

[32] S. Deng, Z. Xiang, J. Taheri, M.A. Khoshkholghi, J. Yin, A.Y. Zomaya, S. Dustdar, Optimal application deployment in resource constrained distributed edges, IEEE Trans. Mobile Comput. 20 (5) (2021) 1907–1923, https://doi.org/10.1109/TMC.2020.2970698.

[33] J. Yan, S. Bi, Y.J. Zhang, M. Tao, Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency, IEEE Trans. Wireless Commun. 19 (1) (2020) 235–250, https://doi.org/10.1109/TWC.2019.2943563.

[34] C. Shu, Z. Zhao, Y. Han, G. Min, H. Duan, Multi-user offloading for edge computing networks: a dependency-aware and latency-optimal approach, IEEE Internet Things J. 7 (3) (2020) 1678–1689, https://doi.org/10.1109/JIOT.2019.2943373.

[35] L. Yang, H. Zhang, X. Li, H. Ji, V.C.M. Leung, A distributed computation offloading strategy in small-cell networks integrated with mobile edge computing, IEEE/ACM Trans. Netw. 26 (6) (2018) 2762–2773, https://doi.org/10.1109/TNET.2018.2876941.