A Cooperative Partial Computation Offloading Scheme for Mobile Edge Computing Enabled Internet of Things

Zhaolong Ning, *Member, IEEE*, Peiran Dong, Xiangjie Kong, *Senior Member, IEEE*, and Feng Xia, *Senior Member, IEEE*

***Abstract*—With the evolutionary development of latency sensitive applications, delay restriction is becoming an obstacle to run sophisticated applications on mobile devices. Partial computation offloading is promising to enable these applications to execute on mobile user equipments with low latency. However, most of the existing researches focus on either cloud computing or mobile edge computing (MEC) to offload tasks. In this paper, we comprehensively consider both of them and it is an early effort to study the cooperation of cloud computing and MEC in Internet of Things. We start from the single user computation offloading problem, where the MEC resources are not constrained. It can be solved by the branch and bound algorithm. Later on, the multiuser computation offloading problem is formulated as a mixed integer linear programming problem by considering resource competition among mobile users, which is NP-hard. Due to the computation complexity of the formulated problem, we design an iterative heuristic MEC resource allocation algorithm to make the offloading decision dynamically. Simulation results demonstrate that our algorithm outperforms the existing schemes in terms of execution latency and offloading efficiency.**

***Index Terms*—Internet of Things (IoT), mobile edge computing (MEC), partial computation offloading, resource allocation.**

I. Introduction

MOBILE user equipments (UEs), including smart phones and laptops, have recently set off a new wave with the evolution of Internet of Things (IoT). Traditional applications are difficult to meet the increasing requirements of IoT-based UEs, such as quality of service (QoS). Recently, a number of novel applications emerge and are quickly favored by users, e.g., image identification, online games, augmented reality, and Internet of Vehicles [1], [2]. Although CPU becomes more and more powerful nowadays, applications running on mobile devices still intend to offload full or partial computation tasks due to their limited computing or hardware capabilities [3]. Strict delay restrictions have become an obstacle to run sophisticated applications on mobile devices [4], with the result that UEs cannot handle large amounts of computing tasks in a short period of time. Moreover, we cannot only rely on programmers to do endless code optimization. Consequently, computation offloading is one of the most efficient way to reduce the execution delay of applications.

Mobile cloud computing (MCC) used to be viewed as a promising approach to offload the computation tasks, since rich computing resources can significantly reduce application processing latency. However, task offloading to the cloud server, spatially far from UEs, can cause high transmission delays. With existing infrastructures, we can hardly reduce network delays between servers and base stations. To overcome these limitations, a new technology, called mobile edge computing (MEC), is advocated. Compared with MCC, MEC has the following characteristics.

1) *Closeness:* Edge computing occurs near data sources. The key idea of MEC is to bring the computing and storage resources of central servers to the network edge as well as in the proximity of users [5].
2) *Diversity:* More than roadside units (RSUs), any computing or networking resource, such as vehicles, can be an edge device, whose computing capability is also distinct.
3) *Resource-Constrained:* Computational resources for MEC within a particular area are limited, and edge nodes or servers generally have lower computing capability than cloud servers.

When the offloading technique is applied, computation partitioning problem is one of the fundamental issues [6]. Consider that the application consists of a series of modules, thus the aim of computation partitioning is to decide which module to be offloaded and how to be executed (locally or remotely), such that the overhead can be minimized. To conduct such an offloading decision, there are several challenges as follows.

1) Traditional offloading schemes are based on two kinds of entities: a) the UEs and the cloud server or b) the UEs

Manuscript received June 17, 2018; revised August 11, 2018; accepted August 28, 2018. Date of publication September 4, 2018; date of current version June 19, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61572106 and Grant 61502075, in part by the Fundamental Research Funds for the Central University under Grant DUT18JC09 and Grant DUT17RC(4)49, in part by the Dalian Science and Technology Innovation Fund under Grant 2018J12GX048, and in part by the State Key Laboratory for Novel Software Technology, Nanjing University under Grant KFKT2018B04. *(Corresponding author: Feng Xia.)*

Z. Ning is with the Key Laboratory for Ubiquitous Network and Service Software of Liaoning Province, School of Software, Dalian University of Technology, Dalian 116620, China, and also with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210008, China (e-mail: f.xia@ieee.org).

P. Dong, X. Kong, and F. Xia are with the Key Laboratory for Ubiquitous Network and Service Software of Liaoning Province, School of Software, Dalian University of Technology, Dalian 116620, China.

Digital Object Identifier 10.1109/JIOT.2018.2868616

and the MEC server. Few studies consider the synergy between MCC and MEC, which can not only balance the load of servers but also guarantee the QoS of IoT UEs.

2) Studies in recent years have mostly focused on single-user offloading problems. However, in a multiuser environment, the user's decision depends not only on the overhead it saves but also the interference among users.
3) To optimize the execution delay or battery energy consumption of the application, the offloading model is often constructed as a user independent offloading model. However, users interact with each other and compete for computing resources. Under the premise of the multiuser situation, computing resource restriction should be taken into account.

Although both MCC and MEC are able to realize time-saving and energy-efficient traffic offloading, few existing researches study their cooperation. In this paper, we demonstrate that MCC and MEC are suitable for different scenarios and they are complementary with each other. When the number of IoT-based UEs is no more than twice of the number of MEC servers, MEC provides better service than MCC with its advantage of short transmission delay. However, if the number of users increases, the advantage of rich computational resources in MCC is highlighted.

We start from the single-user computation offloading problem, which can be solved by the branch and bound algorithm. Then we extend it to the multiuser model, which is formulated as a mixed integer linear programming (MILP) problem. By taking both MEC resource restriction and interference among IoT-based UEs into consideration, the multiuser computation offloading problem (MCOP) is much more complicated than the existing offloading problems. Since the formulated MILP problem is NP-hard, we design an iterative heuristic MEC resource allocation (IHRA) algorithm for the MCOP. The average execution time, including data transmission time and processing time, is the optimization target, because it is the most significant indicator for high computation demanding and latency sensitive applications. IHRA can obtain the optimal offloading decision with low computational complexity, so that the average execution latency can be largely reduced. The main contributions of this paper are as follows.

1) To the best of our knowledge, this paper is an early effort to integrate MCC and MEC for computation offloading. We comprehensively consider the characteristics of the abundant computing resources in MCC and the low transmission delay in MEC, to ensure that the designed offloading framework is suitable for different network scenarios.
2) We model the single user computation offloading problem (SCOP) as an MILP problem at first. Then in multiuser situation, both MEC resource constraint and interference among multiple users are taken into account. The MCOP is formulated as an MILP problem, which is NP-hard to solve.
3) Due to the complexity of the formulated problem, we design an IHRA algorithm to solve the MCOP. Its computational complexity is $\mathcal{O}(N(M + \log_2 N))$, where M and N are the number of servers and UEs, respectively.
4) Performance evaluation demonstrates the effectiveness of our solution in terms of execution latency and offloading efficiency. In order to utilize network resources efficiently, we can obtain a suitable ratio of the number of MEC servers and users according to the simulation results, i.e., the number of the MEC servers is half of the number of IoT UEs.

The rest of this paper is organized as follows. In Section II, we review the related work. We specify the system model and problem formulation in Section III. Section IV describes the proposed IHRA algorithm for the MCOP. Performance evaluations are illustrated in Section V, and Section VI concludes this paper.

II. Related Work

The computation offloading and resource allocation attract the attention of many researchers in recent years. The execution latency [7], [8] and energy assumption [9], [10] are usually two criterions for offloading performance evaluation.

A. Full Offloading

Liu *et al.* [7] proposed a 1-D search algorithm to minimize the execution delay, which takes application buffer queueing state and available processing powers into account. Dynamic voltage and frequency scaling and energy harvesting techniques are utilized in [8] to optimize the data transmission for computation offloading, which can also reduce the failure of offloading applications. Wang *et al.* [11] constructed a three-layer traffic system to minimize the average offloading response time. Queueing theory is applied to model the moving vehicle-based edge nodes. Labidi *et al.* [12] designed a jointly optimizing scheduling and offloading strategy to promote quality of experience (QoE). With the average execution delay constraint, the energy consumption can be largely reduced. However, most of the current studies merely consider the single user situation [13], [14].

In [15], all UEs are divided into two groups according to the amount of data that they need to offload. The UEs in the first group are allowed to access the MEC server, while the second group can only process tasks locally. The optimal transmission power can be determined according to the allocation of communication and computation resources. Hou *et al.* [16] focused on reducing the saturated backhual bandwidth by collaborative edge computing in wireless mesh networks.

B. Partial Offloading

Zhao *et al.* [17] formulated the partial offloading problem as a nonlinear constraint problem and solve it by linear programming. In [18], an optimal resource allocation policy is designed by sorting the application latency constraints in a time division multiple access-based system. Wang *et al.* [19] presented an energy-optimal partial offloading scheme, depending on the maximal application latency constraint. In [10], an iterative

algorithm is proposed to find the optimal value of the transmission rate in uplink. Mao *et al.* [20] formulated the energy consumption minimization problem by taking the application buffer into account. An online algorithm based on Lyapunov optimization is proposed to decide the optimal CPU frequency and transmission task, as well as allocate the bandwidth resources. In [21], a resource block scheduling problem is formulated in narrowband-IoT. The heuristic algorithm considers both power control and relay selection.

C. Collaboration of MCC and MEC

MCOP is more common and complex than single user problem. In [19], multiuser computation partitioning problem is formulated as an MILP problem and solved by a greedy heuristic algorithm. The shortcoming of this paper is that it neither considers the collaboration of MCC and MEC, nor takes the interference among users into consideration. Zhang *et al.* [22] took the interference among users into consideration. However, it focuses on full offloading and does not consider the collaboration of MCC and MEC. Different from the studies above, this paper comprehensively considers the cooperation of MCC and MEC, where IoT-based UEs can either offload their tasks through MCC or MEC depending on the execution latency.

III. System Model and Problem Formulation

This section mainly specifies the system model and formulates the investigated problem. The system model includes the application model and the computation offloading framework model. The problem description begins with a single user and extends to multiple users.

A. System Model

There are various mobile applications emerging in the past few years. We mainly focus on computation-intensive and delay-sensitive applications, which require high computational capability as well as low execution latency to satisfy user QoE. IoT-based UEs with low latency requirement applications, such as applications based on augmented or virtual reality, online gaming or remote desktop, may profit from computation offloading to the cloud or MEC [23].

Basically, one UE consists of a code analyzer and a system manager [24]. The code analyzer is responsible to determine which part of the application can be offloaded, depending on application types and equipment hardware conditions. The main factor affecting whether applications should be all offloaded or partial offloaded is the type of application. For example, if the application consists of modules whose input data is private information of users, such as the name and ID card, then it should be partial offloaded. Otherwise, it can be all offloaded. The system manager is in charge of monitoring various parameters, including available bandwidth, computation capability, the offloaded data size, and UE's transmission power. Computation partitioning, also known as partial offloading, is a complex and challenging process, which needs to consider many factors, such as users' privacy, communication link quality, UE's capabilities, cloud capabilities,

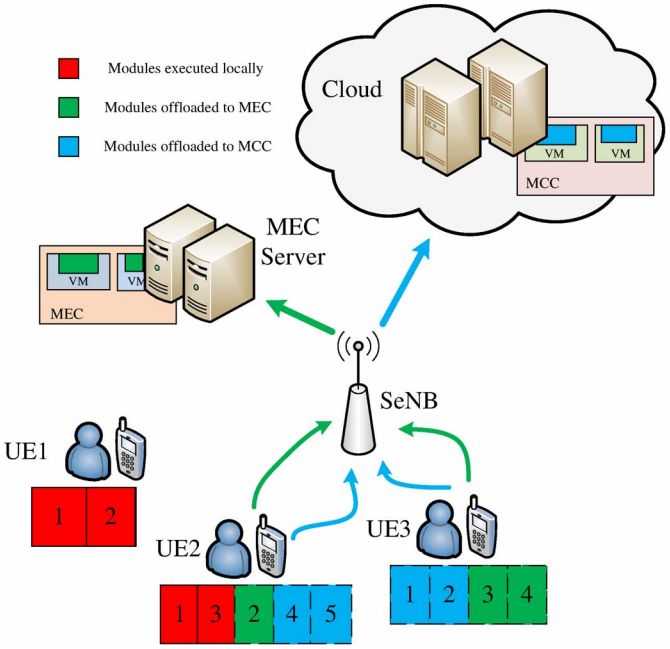


Fig. 1. Partial computation offloading example.

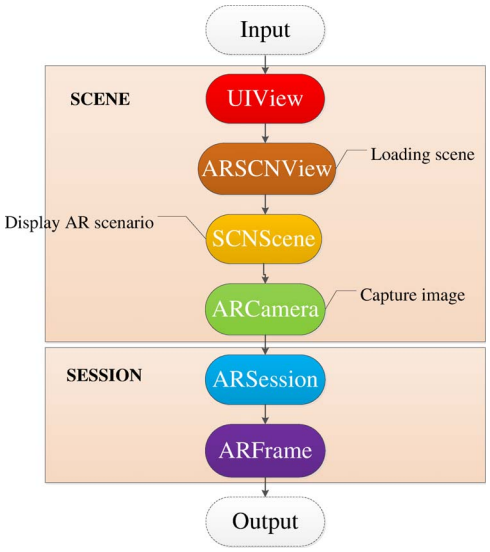


Fig. 2. Application of augmented reality framework ARkit.

and resource availability [25]. Generally, a significant factor, influencing the decision of either full or partial offloading, is the ability of offloading [23]. If the application is composed of nonoffloadable parts (e.g., user input data and position information), those parts can only be executed locally on UEs while the rest of the application can choose either to be offloaded to the remote server or not. An illustrative example is shown in Fig. 1, and the whole application is executed locally on UE1. UE2 offloads its second module to the MEC server, and the fourth together with the fifth modules are offloaded to the cloud server, while the rest is processed by the local device. The first and second half of the application are offloaded by MEC and MCC, respectively.

All modules of the application can be either independent or interdependent. In this paper, we simplify the complex application module dependency system into a linear sequence processing module, as shown in Fig. 2. It is the application of augmented reality framework ARkit. First, augmented reality application is a typical complex and delay-sensitive application. Second, the ARkit framework can be divided into several independent submodules. The above two points are in line with the research conditions of this paper. So we choose the ARkit framework. Moreover, the proposed model is still applicable when extending to other applications whose submodules are linear independent. We can simulate these applications by adjusting the number of submodules and the amount of offloaded data. Nonlinear applications with complex internal dependencies are beyond the scope of this paper. In Fig. 2, arrows reflect the dependencies among modules. The output of the previous module is the input of the next module. Each module can be executed on three entities, i.e., UEs, RSUs, and cloud. For an application, the very first input data is initiated on the local device. Thus the input data of the first submodule comes from UEs. However, the first submodule itself does not have to be executed locally. Similarly, when the entire application is completed, the very last output data should be transmitted back to the local device. In order to facilitate the formula description, two additional virtual modules are added to the framework. Since the consumption of computation energy is far less important than execution latency in terms of delay-sensitive applications, the performance indicator is the latency. It represents the whole execution time of the application, including all modules' computation time and data transmission time among the modules executed on different platforms. Although the ARkit does not have nonoffloadable parts, the proposed model is still suitable for those applications with nonoffloable parts. For those nonoffloadable submodules, we can manually initialize it to the form of $y_{i,j,\alpha} = 1$ in the simulation, which represents that they are executed locally.

B. Single User Computation Offloading

We start from the SCOP, in which only one user makes the computation offloading request. As mentioned previously, we model the application of IoT-based UEs as a linear sequence processing program, which includes η modules. Each module can choose to be either offloaded to the remote server or processed locally on UEs. The processing time of module j by local processing, edge offloading, and cloud offloading are p_j^L, p_j^E, and p_j^C, respectively, where $p_j^L > p_j^E > p_j^C$. The data transmission time between two adjacent modules j-1 and j is t_j, if the two modules are processed on different platforms; otherwise, the transmission time can be ignored. Notice that the very first input/last output data should be from/to the local UE. Thus, two additional modules 0 and $\eta + 1$ are added to the linear sequence model as virtual input and output modules for the application.

Given the computation cost p_j $(1 \leqslant j \leqslant \eta)$ and data transmission cost t_j $(0 \leqslant j \leqslant \eta + 1)$, the SCOP makes the decision that which modules should be offloaded onto the edge node or the cloud, so that the total execution delay can be minimized.

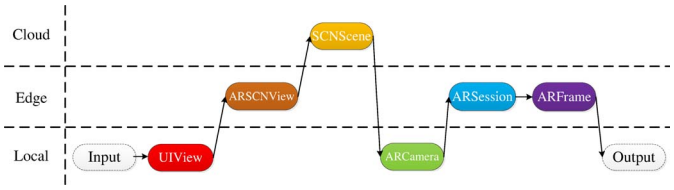


Fig. 3. Example of SCOP offloading schedule.

The investigated problem is formulated as follows:

$$\min \text{Delay} = \sum_{j=1}^{\eta} p_j + \sum_{j=1}^{\eta+1} t_j \tag{1}$$

s.t.

$$p_j = \alpha p_j^L + \beta p_j^E + \gamma p_j^C$$

where $\alpha + \beta + \gamma = 1$, and

$$\alpha = \begin{cases} 1, & \text{if the task is processed locally} \\ 0, & \text{otherwise.} \end{cases}$$

$$\beta = \begin{cases} 1, & \text{if the task is processed by the MEC server} \\ 0, & \text{otherwise.} \end{cases}$$

$$\gamma = \begin{cases} 1, & \text{if the task is processed by the MCC server} \\ 0, & \text{otherwise.} \end{cases}$$

The variable p_j has relationship with data size and CPU processing power, while t_j is affected by the communication environment, such as the channel bandwidth. Both p_j and t_j will be analyzed in detail in Section III-C in conjunction with the multiuser problem.

An illustrative example of SCOP solution is shown in Fig. 3. Six modules of augmented reality application are assigned to the appropriate platform for processing as shown in Fig. 2. Each module receives the output data of the previous module as the input data. The cooperation of MEC and MCC enables latency sensitive applications to run on IoT-based equipments.

C. Multiuser Computation Offloading

In this section, we study the problem of multiuser computation offloading. The multiuser hierarchy computation offloading model is shown in Fig. 4. The IoT-based UEs send offloading requests to the small evolved NodeBs (SeNBs), which make offloading decisions to the MCC server or the MEC server. SeNBs are usually close to users and MEC servers, so the transmission delay is small. Its computing power is also stronger than that of the UE, which can perform the offloading scheduling quickly. If SeNB is overloaded in prosperous areas such as the city center and train stations, we can adopt a joint scheduling scheme for distributed SeNBs. Neighboring SeNBs can be used to complete the excessive scheduling tasks, which can meet users' demands with a little bit performance degradation on the execution delay. Notice that multiuser models are not simply superimposed by single-user models. In SCOP, we do not consider the interference among users, since the whole channel is allocated to the single user without any resource competition. In MCOP, users' competition for channel resources challenge the offloading process. As a result, the total transmission time of multiuser is larger than that of the single user.

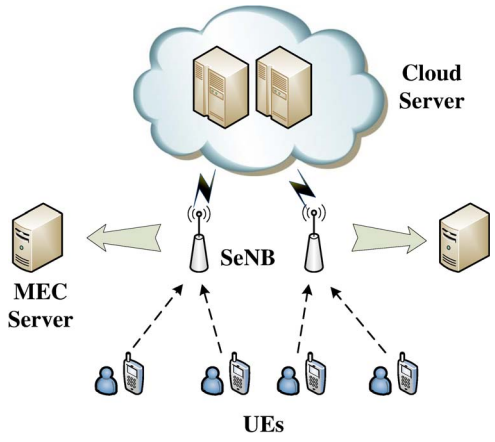


Fig. 4. Multiuser hierarchy computation offloading model.

We consider there are λ user requests within a time slot, and the number of users is relatively stable within each time slot. All IoT-based UEs are in the same frequency band, where the interference among users exists [22]. The total bandwidth B is divided into N channels. Users upload requests and data to the SeNBs by orthogonal frequency-division multiple access, where every user in the same channel is orthogonal to others and each channel can only be allocated to one user. Each computation task $\tau_{i,j} = \{ d_{i,j}, c_{i,j} \}$, where $d_{i,j}$ is the input data size of the ith user's jth module, and $c_{i,j}$ is the required CPU cycles to complete the computation task. For each task, it can be processed either locally on IoT-based devices or remotely on MCC/MEC servers. The total execution time consists of two parts, i.e., processing time $p_{i,j}$ and transmission time $t_{i,j}$.

1) Processing Time: We define f_i^l, f_k^e, and f^c as the computation capability (i.e., CPU-cycle frequency) of IoT-based UEs, MEC sever, and MCC server, respectively. Herein, f_i^l and f_k^e are relevant to the processing power of different users and edge nodes, and f^c is fixed during the computation offloading [10], [19]. When offloaded task $\tau_{i,j}$ is processed locally, the processing time $p_{i,j,0}^L$ is

$$p_{i,j,0}^L = \frac{c_{i,j}}{f_i^l}. \tag{2}$$

When the task is offloaded onto the edge node, the processing time $p_{i,j,k}^E$ is

$$p_{i,j,k}^E = \frac{c_{i,j}}{f_k^e} \tag{3}$$

where k denotes the kth MEC sever ($1 \leqslant k \leqslant M$). When task $\tau_{i,j}$ is processed on the cloud, the processing time $p_{i,j,M+1}^C$ can be calculated by

$$p_{i,j,M+1}^C = \frac{c_{i,j}}{f^c} \tag{4}$$

where the $(M+1)$th server denotes the MCC server. As mentioned before, task $\tau_{i,j}$ is arranged to be processed at one of the three platforms, so that we can calculate the total processing time by

$$p_{i,j} = \alpha p_{i,j,0}^L + \beta p_{i,j,k}^E + \gamma p_{i,j,M+1}^C \tag{5}$$

where $\alpha + \beta + \gamma = 1$.

2) Transmission Time: Generally, users send input data to the edge node or the cloud through SeNBs instead of sending directly, thus SeNBs undertake the task of scheduling in our offloading model. If users intend to make offloading decision by themselves, they need real-time information of the MEC and MCC servers. That is to say, although we can reduce the transmission time of the offloading request, additional transmission delay can be caused. When the SeNB is in the proximity of the MEC server in local regions, the transmission time can be omitted [26], [27]. In addition, due to the fact that the size of output data delivered from the remote server to the local device is much smaller than that of the input data, the time overhead of the backhual link can be ignored [27], [28]. Therefore, we mainly focus on the upload link from the local device to the SeNB and the SeNB to the cloud server.

We define three binary variables $y_{i,j,\alpha}$, $y_{i,j,\beta}$, and $y_{i,j,\gamma}$ to denote whether the jth module of the ith user is executed locally, by the MEC server or by the MCC server, respectively. If user i offloads modules to remote server k through SeNBs on channel n, the achievable transmission rate $r_{i,k,n}$ can be calculated by

$$r_{i,k,n} = \omega \log_2 \left(1 + \frac{p_{i,k,n} h_{i,k,n}}{\sigma^2 + I_{i,k,n}} \right) \tag{6}$$

where ω is the bandwidth. Since total bandwidth B is divided into N channels, $\omega = B/N$. The variable $p_{i,k,n}$ is the transmission power, and $h_{i,k,n}$ is the channel gain representing the link transmission loss from user i to remote server k on channel n. The denominator is signal to interference plus noise ratio, where σ^2 is the noise power and $I_{i,k,n}$ denotes the interference caused by nearby users to user i on channel n. It can be expressed as

$$I_{i,k,n} = \sum_{x=1, x \neq i}^{\lambda} \sum_{y=1}^{M+1} a_{x,y,n} p_{x,y,n} h_{x,y,n}^k \tag{7}$$

where x and y are the serial number of users and MEC servers, respectively. The variable $a_{x,y,n}$ is a binary variable, and when it equals to 1, the channel n is allocated to perform task $\tau_{x,y}$ from UE x to SeNB y. Otherwise, $a_{x,y,n} = 0$. Hence, the total transmission rate of this frequency band can be obtained by

$$r_{i,k} = \sum_{n=1}^{N} a_{i,k,n} r_{i,k,n} \tag{8}$$

where each task can occupy at most one channel, i.e., $\sum_{n=1}^{N} a_{i,k,n} \leq 1$. Therefore, the transmission time of the ith user on module j can be represented as

$$t_{i,j} = t_{i,j,\alpha \to \beta} + t_{i,j,\alpha \to \gamma} + t_{i,j,\beta \to \gamma} + t_{i,j,\gamma \to \beta}. \tag{9}$$

The transmission time consists of four situations, where arrows indicate the direction of computing platforms for offloading, e.g., $t_{i,j,\alpha \to \beta}$ denotes the situation that module $j-1$ is processed locally, and module j is offloaded to the edge node. Thus, we have to consider the transmission time from the local IoT-based device to the remote MEC server.

Equation (10) lists the calculation formulations for the four situations

$$\begin{aligned} t_{i,j,\alpha\to\beta} &= y_{i,j,\beta}\frac{d_{i,j}}{r_{i,k}} \\ t_{i,j,\alpha\to\gamma} &= y_{i,j,\gamma}\left(\frac{d_{i,j}}{r_{i,M+1}}+\pi_{i,j,k}\right) \\ t_{i,j,\beta\to\gamma} &= y_{i,j-1,\beta}y_{i,j,\gamma}\pi_{i,j,k} \\ t_{i,j,\gamma\to\beta} &= y_{i,j-1,\gamma}y_{i,j,\beta}\pi_{i,j,k} \end{aligned} \tag{10}$$

where $\pi_{i,j,k}$ denotes the transmission time from SeNB k to the MCC server.

3) Problem Formulation: As mentioned above, the computation offloading delay includes two parts: a) the processing time and b) the transmission time. There is a tradeoff between the two parts due to their differences in computing capability and distances. The MCOP can be formulated as follows:

$$\min_{\alpha,\beta,\gamma} Delay = \sum_{i=1}^{\lambda}\sum_{j=1}^{\eta}\left(p_{i,j}+t_{i,j}\right) \tag{11}$$

s.t.

$$\begin{aligned} C1:&\quad \alpha+\beta+\gamma=1, \alpha,\beta,\gamma\in\{0,1\} \\ C2:&\quad y_{i,j,\alpha}+y_{i,j,\beta}+y_{i,j,\gamma}=1 \\ &\quad y_{i,j,\alpha}, y_{i,j,\beta}, y_{i,j,\gamma}\in\{0,1\} \\ C3:&\quad \sum_{j} y_{i,j,\alpha}+y_{i,j,\beta}+y_{i,j,\gamma}=\eta \quad \forall i\in[1,\lambda] \\ C4:&\quad \sum_{n=1}^{N} a_{i,k,n}\le 1, a_{i,k,n}\in\{0,1\} \\ C5:&\quad \sum_{i=1}^{\lambda} y_{i,j,\beta}\le \min\{\lambda, M\} \quad \forall j\in[1,\eta] \\ C6:&\quad 0\le\sum_{i,j} y_{i,j,\beta}\le M \\ C7:&\quad \sum_{i=1}^{\lambda} y_{i,j,\gamma}\le\lambda \quad \forall j\in[1,\eta] \\ C8:&\quad 0\le\sum_{i,j} y_{i,j,\gamma}\le\lambda\eta \\ &\quad (2)\text{–}(10) \end{aligned}$$

where constraints C1 and C2 guarantee that each module can only be processed on one platform, i.e., either the local IoT-based device, the MEC server or the cloud. Constraint C3 ensures all application modules of user i can be processed. Constraint C4 indicates each user can only be assigned one channel. Constraints C5 and C6 reflect that MEC resources of MEC are limited so that each MEC server can only handle one module offloading request. The equation $y_{i,j,\beta}=1$ means the jth module of the ith user is executed by MEC servers. Meanwhile, the MEC server is constraint in the MCOP, and its total number is M. The number of MEC servers occupied by users cannot exceed the total number of MEC servers. Therefore, the parameter in constraint 5 has to be less than M. Instead, Constraints C7 and C8 illustrate that cloud resources are not limited, which can be used by multiple users simultaneously.

The multiuser computation partitioning for the offloading strategy between the local device and the remote cloud server is an MILP problem, which is proved to be NP-hard [6]. Taking the interference among users and the computing resource restriction into consideration, our MCOP is more complicated and difficult to solve than the multiuser computation partitioning problem. Thus, we design a heuristic algorithm to solve MCOP. For each UE, we first simplify MCOP to SCOP, and obtain the initial optimal solution by the branch and bound algorithm, where the MEC resources are unconstrained. Then, the initial offloading schedule of UEs with resource conflicts are adjusted due to the limited MEC servers.

IV. Iterative Heuristic Resource Allocation Algorithm

We specify the IHRA algorithm in this section. First, some important data structures and functions are illustrated before presenting the IHRA algorithm.

A. Initial Input

A 3-D decision matrix φ, whose shape is $3\lambda(\eta+2)$, is created to record the offloading decision of IoT-based UEs. Herein, 3 indicates three computing platforms for offloading: 1) the local IoT-based device; 2) the MEC server; and 3) the MCC server, $\eta+2$ and λ represent the number of modules and users, respectively. Note that we add two additional modules 0 and $\eta+1$ as input and output modules, which virtually exist on UEs. Thus, the size of the second dimension of the decision matrix is $\eta+2$. For example, $\varphi[1;5;6]=1$ means the fifth module of the sixth UE is offloaded onto the MEC server. According to Constraint C2, $\varphi[0;5;6]$ and $\varphi[2;5;6]$ must be 0 since each module can only run on one platform.

Before solving the MCOP, we first recall the SCOP, which is the basis of our heuristic algorithm. When the computation-intensive task is executed on IoT-based UE i, it may need offloading services. Consequently, UE i sends its offloading requirement set $\phi_i=\{D_i, C_i, f_i^l, p_i\}$ to the nearest SeNB, in which the D_i, C_i, f_i^l, and p_i represent the data volume of all modules, the required CPU cycles, the computation capability of the device and the transmission power, respectively. Once the SeNB receives the request, it solves the SCOP by using the branch and bound method and obtains an initial decision matrix.

B. Feedback Function

Since the MEC resource is limited under the multiuser condition, we have to deal with the situation of resource conflicts in the initial decision matrix that is calculated by solving SCOP. The feedback function F is designed to alleviate the conflicts and allocate the MEC resource, which can be defined by

$$F = D_{\text{orig}} - D_{\text{adj}} \tag{12}$$

where D_{orig} is the original total execution delay calculated by SCOP, and D_{adj} is the adjusted total execution delay caused by modification.

Algorithm 1 IHRA for MCOP

Input: λ, M, $\phi_i = \{ D_i, C_i, f_i^l, p_i \}$, f^e and f^c;
Output: The offloading decision matrix φ;
1: Compute the execution delay D_{orig};
2: Obtain initial offloading decision matrix $\varphi_{initial}$;
3: Count $\tilde{\lambda}$
4: Compute the adjusted execution delay D_{adj};
5: Obtain adjusted offloading schedule $\varphi_{adjusted}$;
6: $F = D_{orig} - D_{adj}$
7: Sort list F in descending order
8: **while** $\tilde{\lambda} > M$ **do**
9: Select the first UE λ_i from the list F
10: Update $\varphi_{initial}[\lambda_i] := \varphi_{adjusted}[\lambda_i]$
11: Update $D_{orig}[\lambda_i] := D_{adj}[\lambda_i]$
12: Remove λ_i from $\tilde{\lambda}$
13: $\tilde{\lambda} \Leftarrow \tilde{\lambda} - 1$
14: **end while**
15: **return** final offloading decision matrix φ;

In order to minimize the overall execution delay, we prefer to adjust the UE with the least impact on the entire system. Before and after the modification, the first UE in the feedback function has the smallest overall delay increase. In other words, this UE has the least dependence on the MEC resources.

C. Algorithm Description

Algorithm 1 presents the pseudo code of the IHRA scheme. The input of IHRA includes λ UEs with their offloading requirements ϕ, M edge nodes with their computation capabilities f^e, and the computation capability of the cloud server f^c. First, we compute the optimal application execution delay D_{orig} for each UE without considering the MEC resource constraint. The branch and bound algorithm is adopted to solve the SCOP, which outputs the initial offloading schedule recorded in matrix φ (lines 1 and 2). Then, we count the UEs that occupy the MEC resources, and denote it as $\tilde{\lambda}$. Whether applications on UE i are offloaded onto the edge node can be obtained by checking the second row of UE i's decision matrix (line 3). The adjusted execution delay D_{adj} and offloading decision matrix are computed similar with SCOP but without considering MEC resources. To reduce the time complexity of our algorithm, only the UEs in $\tilde{\lambda}$ are computed (lines 4 and 5). After obtaining D_{orig} and D_{adj}, we can compute the feedback function according to (12) (line 6). Next, a while loop is formulated to iteratively perform MEC resource allocation by updating the initial decision matrix until all MEC resource conflicts are solved (lines 8–14). In each iteration, we sort list F in a descending order. The first UE λ_i in F is the current adjustment target, because our method has the minimal average delay growth when adjusting the offloading schedule of λ_i. The offloading schedule of λ_i can be found in the adjusted decision matrix. The value of λ_i in φ_{initial} and D_{orig} is updated with $\varphi_{\text{adjusted}}$ and D_{adj}, respectively. We can obtain the final offloading decision matrix φ after the resource allocation (line 15).

For example, if there are five MEC servers in one local region, the IoT-based UEs from λ_1 to λ_{10} send their offloading requirements to the SeNBs, which compute the values of D_{orig} and φ_{initial}. UE λ_3 does not occupy MEC resources, i.e., all offloaded requests of λ_3 are processed by the cloud server. Then the SeNB computes the D_{adj} and $\varphi_{\text{adjusted}}$ of the rest of the UEs except λ_3. Thus, the number of UEs in $\tilde{\lambda}$ is 9 and feedback function list F can be obtained. Since only five UEs can be served by MEC servers, the offloading requirements of four UEs have to be adjusted from the MEC servers to the cloud server. We adjust the offloading decision according to the list F, which is sorted in a descending order. UE λ_5 ranks the first one in the list, i.e., when we adjust the offloading schedule of λ_5, the average delay growth of all UEs is minimal. Thus, its offloading tasks are adjusted to be processed by the cloud server instead of the MEC server. The decision matrix and execution delay are updated. The similar process is repeated four times until no MEC resource conflict exists. The IHRA algorithm outputs the final offloading decision.

D. Computation Complexity Analysis

In this section, we prove that the proposed IHRA algorithm can be resolved in polynomial time.

Theorem 1: The computation complexity of the proposed IHRA algorithm is $\mathcal{O}(N(M + \log_2 N))$, where N is a constant multiple of UE number λ, M is the number of MEC servers, and $\lambda \geqslant M$ holds.

Proof: The time complexity of IHRA mainly contains two parts, i.e., the initialization stage and the while loop stage. For the former part, the computation complexity of the branch and bound algorithm to solve SCOP is $\mathcal{O}(\lambda(M+2)3^{\eta-1})$. The parameters λ, η, and M remain unchanged during the computation of the adjusted execution delay and offloading schedule, and its time complexity is $\mathcal{O}(\lambda 2^{\eta})$. The time complexity of the sorting process is $\mathcal{O}(\lambda \log_2 \lambda)$. Thus, the time complexity of the initialization stage is $\mathcal{O}(\lambda((M + 2)3^{\eta-1} + 2^{\eta} + \log_2 \lambda))$. In the while loop, it mainly performs some update operations, whose time complexity is $\mathcal{O}(\lambda - M)$. We consider that the number of UEs is much larger than that of MEC servers, i.e., $\lambda \geqslant M$. If $\lambda < M$, it does not need to perform the while loop to handle MEC resource conflicts. By combining two parts of IHRA algorithm, the total time complexity can be expressed as $\mathcal{O}(\lambda((M+2)3^{\eta-1} + 2^{\eta} + \log_2 \lambda) + (\lambda - M))$ if $\lambda \geqslant M$, and $\mathcal{O}(\lambda((M + 2)3^{\eta-1} + 2^{\eta} + \log_2 \lambda))$ if $\lambda < M$. Because η is a constant, $\mathcal{O}(\lambda((M+2)3^{\eta-1}+2^{\eta}+\log_2 \lambda)+(\lambda-M))$ can be simplified as $\mathcal{O}(N(M+\log_2 N))$, where N is a constant multiple of λ. Thus, the proposed algorithm can be solved in polynomial time. ■

Since the computation complexity of the brute-force searching algorithm can be denoted by $\mathcal{O}(\lambda 3^{\eta-1}(M + 2)!)$, the computation complexity of IHRA algorithm is much lower compared with the brute-force searching algorithm. For example, when $\lambda = 40$, $\eta = 6$, and $M = 20$, the time complexity of the brute-force searching algorithm is 5.04×10^{19} times higher than that of our proposed algorithm.

TABLE I
SIMULATION PARAMETERS

Simulation Parameter	Value
Number of application modules η	6
Cloud server CPU frequency f^c	64 GHz
MEC server CPU frequency f^e	16 GHz
UE CPU frequency f^l	1.2 GHZ
Data size of each module $d_{i,j}$	500-1500 KB
Required CPU cycles $c_{i,j}$	0.2-0.3 GHz
Transmit power of UE $p_{i,k,n}$	23 dBm
Noise power σ	-114 dBm

V. PERFORMANCE EVALUATION

In this section, we evaluate the proposed IHRA algorithm based on the layered heterogeneous network. The features of both MCC and MEC are taken into consideration. In the following, we first introduce the simulation setup, and then present the experimental results.

A. Simulation Setup

We select the application of augmented reality framework ARkit as shown in Fig. 2 in our evaluation. The ARkit has six modules, i.e., the number of the application modules is $\eta = 6$. The cloud server is far from UEs, and its computation capability f^c represented by CPU frequency is 64 GHz. The MEC server locates in proximity of the SeNB, whose CPU frequency is 16 GHz. The characteristics of MEC and MCC are distinguished by their CPU frequencies and distances from the user. Each UE has different tasks to be executed. Based on the framework of augmented reality, all applications in our experiment can be divided into several submodules. Different applications can be represented by different volumes of data and required CPU cycles of submodules. In our experiment, the data size and required CPU cycles of each module are randomly generated between [500, 1500] kB and [0.2, 0.3] GHz, which simulate scenarios for different applications. The CPU frequency of UEs is 1.2 GHz. For simplicity, we assume that the same application runs on all UEs. However, we can easily extend the model by considering different users with various applications. The algorithm in Section IV is still able to be applied to solve the offloading decision problem by setting different η in each iteration. There are ten channels for each UE and the bandwidth is 0.2 MHz. The transmission power of each UE is 23 dBm. The noise power is -114 dBm. In addition to the parameter settings listed in Table I, it is also important to initialize different applications. Especially, for those partial offloading applications, their nonoffloadable submodules are manually set to be executed locally, whose indicator $y_{i,j,\alpha}$ is equal to 1. For convenience, the selected ARkit framework does not have nonoffloadable parts.

We compare the proposed IHRA algorithm with three baseline algorithms: 1) MCC-based offloading; 2) MEC-based offloading; and 3) all local-based offloading. MCC-based offloading and MEC-based offloading represent that there are only cloud computing resources and MEC resources, respectively; *all local-based offloading* indicates that all modules are executed on the UEs. Because execution delay is the optimization target considered in this paper, for intuitive performance analysis, we define the average delay ratio (ADR) by

$$\text{ADR} = \frac{D_{\text{IHRA}}}{D_{\text{adj}}}. \tag{13}$$

The numerator is the optimal execution delay of all UEs computed by the proposed IHRA algorithm. The denominator is the optimal execution delay computed by existing schemes. Corresponding to the three basic algorithms: 1) MCC; 2) MEC; and 3) all local-based offloading, D_{adj} is computed in the situation of non-MEC, non-MCC, or all executed locally, respectively. In particular, most works are focusing on the situations of MCC-based offloading and MEC-based offloading.

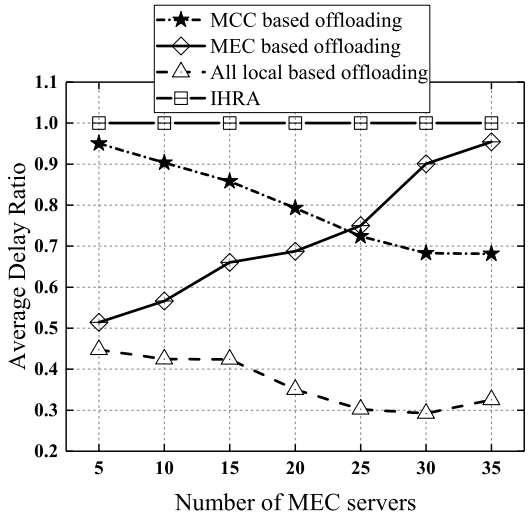


Fig. 5. Comparison of ADR under different number of MEC servers ($\lambda = 40$).

B. Experimental Results

Network performances of the IHRA algorithm are shown in Figs. 5–10.

In Fig. 5, the number of IoT-based UEs is fixed at 40. We compare the ADR-based performance under different number of MEC servers. Among three kinds of ADR curves, the MCC-based offloading and *all local-based offloading* decrease as the number of M increases. When $M \leqslant 15$, the MEC server does not cope with the offloading task, and the addition of the MEC servers reduces the delay by approximately 10% compared to the cloud computing only. From the *all local-based offloading* curve we can discover, computation offloading significantly reduces the execution latency of high demanding applications by approximate 60%. When the value of M is larger than 20, the proposed algorithm achieves a much better performance than the MCC-based offloading, because the MEC processes most of the offloading requests. The total execution delay can be reduced by more than 30%. Furthermore, offloading has a great impact on users, with which the delay of *all local-based offloading* computing is reduced by 70%. Different from the

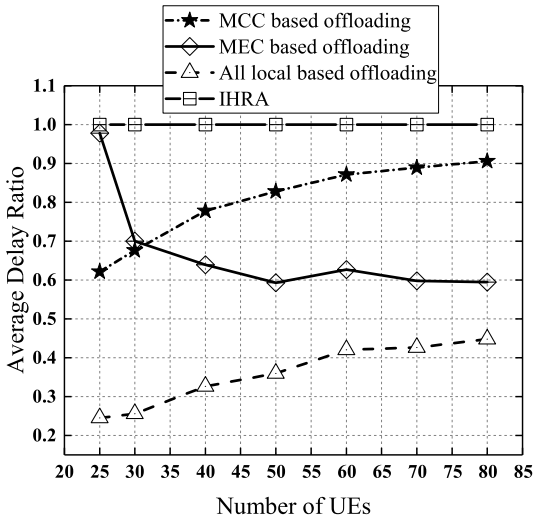


Fig. 6. Comparison of ADR under different number of UEs ($M = 20$).

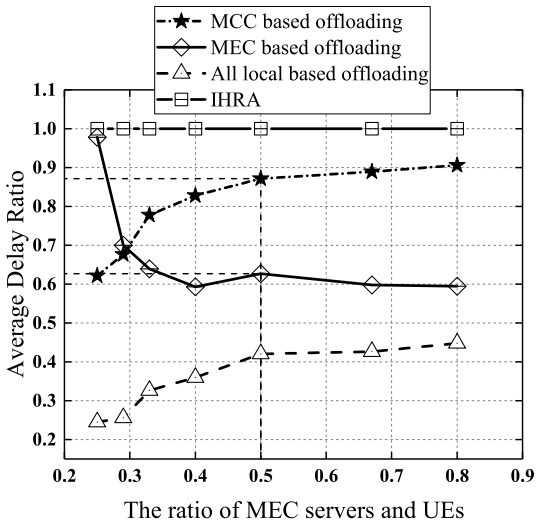


Fig. 7. ADR performance varies from M/λ.

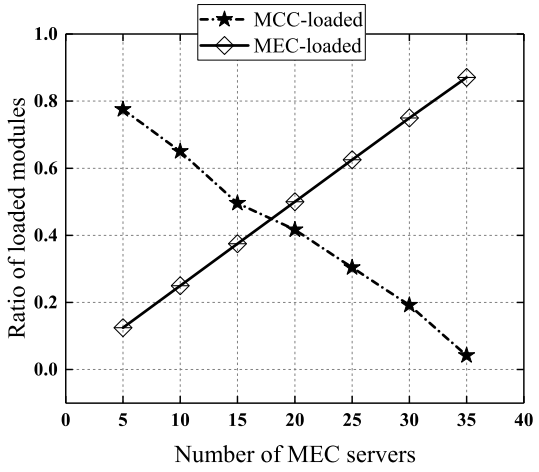


Fig. 8. Proportion of MEC and MCC varies from M.

other two curves, the performance of MEC-based offloading increases as M increases. It reflects the proportion of collaborative cooperation between MEC and cloud computing. When M is large, the MEC resources are adequate, and the delay of the iterative heuristic algorithm can only decrease less than 10%. However, when MEC resources are scarce, our algorithm can reduce the execution delay by 50%, which demonstrates the importance of cloud computing when the MEC server is not enough to handle all requests. Note that the ADR is the indicator of the performance evaluation, which is defined as the ratio of the proposed IHRA algorithm and benchmark methods. The upper bound of the ADR is IHRA. Thus, the performance of the proposed algorithm is always equal to 1 in our simulation. In summary, Fig. 5 illustrates the iterative algorithm provides better performance than other baseline algorithms and is able to provide a better performance by the full utilization of MEC resources.

In Fig. 6, we fix the number of MEC servers to 20, and compare the ADR-based performance of the three baseline algorithms when the number of IoT-based UEs increases from 20 to 85. The performance of the MEC-based offloading

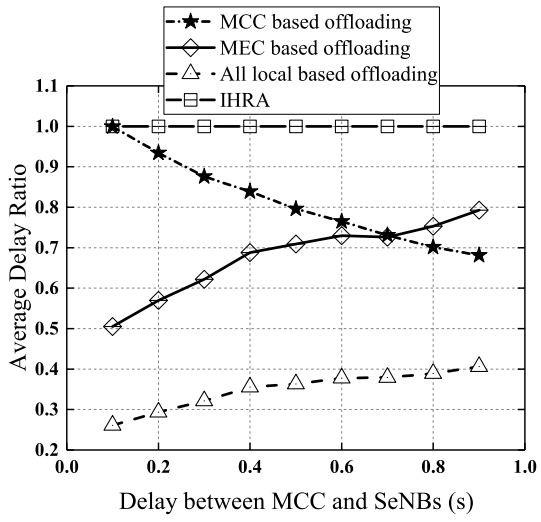


Fig. 9. ADR performance varies from the latency between the cloud server and SeNBs.

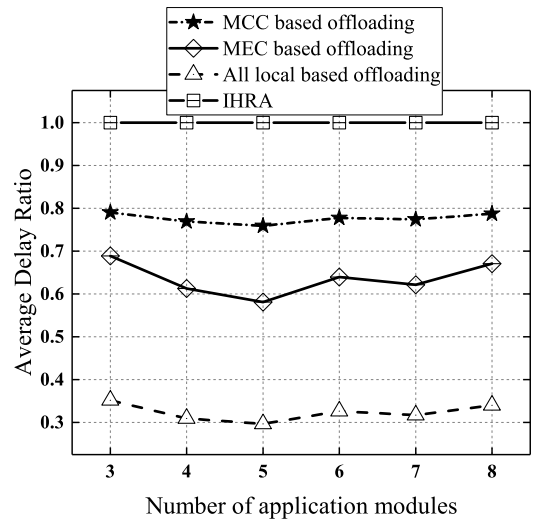


Fig. 10. ADR performance varies from the number of application modules.

decreases as λ increases, because limited MEC resources cannot handle a large number of UEs' offloading tasks. In contrast, when the number of UEs is small, the performance of MCC-based offloading is poor, and it increases as λ adds. This phenomenon shows that the advantage of unconstrained cloud computing resources is obvious in dealing with a large number of user requests. MEC can largely reduce execution delay when resources are relatively sufficient. Since the *all local-based offloading* curve is always under 0.5, whenever the whole application is executed locally, the total delay is always much larger than that of the computation offloading.

The ADR performance varying from the ratio of MEC server number and IoT-based UE number is presented in Fig. 7. Since MEC resources are relatively scarce, resources need to be configured according to the number of users. In contrast, MCC servers have strong computing capability and relatively rich resources, and generally do not consider resource allocation. Therefore, we consider the ratio of MEC servers and UEs, and do not consider the ratio of MCC servers and UEs. It is obvious that the greater the value of M/λ is, the better IHRA performance is. However, considering reasonable and efficient usage of resources and economic costs, we can never infinitely increase MEC resources. We can observe that when $M/\lambda < 1/2$, ADR performance increases rapidly with the increasing of M/λ. At this point, MEC resources are relatively scarce, and more resources are needed. When $M/\lambda > 1/2$, ADR performance increases slowly with the increase of M/λ. MEC resources are gradually becoming saturated at this time. Deploying a lot of MEC servers can reduce the usage efficiency and increase the economic burden. Thus, making the number of the MEC servers equal to half the number of UEs is an effective and economic efficient solution in our investigated problem. The obtained optimal proportion works for our simulations, where the number of UEs and MEC servers is relatively small. In addition, the proposed method is also applicable to a realistic scenario with a large number of users.

Fig. 8 shows the proportion of MEC and MCC loaded modules under different numbers of MEC servers. With the increasing number of MEC servers, the ratios of MEC-loaded and MCC-loaded modules approximately grow and decline linearly, respectively. When the number of MEC servers is relatively small, cloud computing can handle most of the offloading tasks. MCC is indispensable in our offloading model. However, when the MEC resources are sufficient, cloud computing only plays an auxiliary role, which can be viewed as an integral part of the entire offloading system. Without cloud computing, we cannot handle user requests with explosive growth in a short period of time. Therefore, MCC and MEC are complementary with each other, which reflects the cooperative relationship between MEC and MCC.

The latency between the cloud server and SeNBs affects the offloading schedule of the IoT-based UEs. Compared with MEC, the advantages of MCC are with rich computing resources and powerful computing capability. High transmission delay can weaken this advantage since the saved processing delay cannot make up for the increased transmission delay. Fig. 9 presents the ADR performance varying from the latency between the cloud server and SeNBs. Not surprisingly, the performance of MCC-based offloading decreases when the latency increases, and MEC-based offloading reverses. When the latency is large, the performance improvement of the MEC-based offloading algorithm fully demonstrates its advantage of the proximity to users. As for *all local-based offloading*, the difference of latency has little effect on its execution delay.

As shown in Fig. 10, we evaluate the proposed IHRA algorithm under different number of application modules. The number of IoT-based UEs λ and MEC servers M is set to 40 and 20, respectively. The ADR performances of three baseline algorithms remain stable with the changing of module number, which proves that IHRA is suitable for applications with different network situations.

VI. Conclusion

In this paper, we devote to enable the latency sensitive applications to run on IoT UEs by implementing partial computation offloading. First, we consider the single user offloading problem, where the MEC resources are considered to be unconstrained. Later, we extend it to the multiuser offloading problem, where both the resource constraint and the interference among multiple users are taken into consideration. Due to the computation complexity of the formulated problem, an IHRA scheme is put forward for making the computation offloading decision under the multiuser situation. Simulation experiment illustrates the superiority of the proposed algorithm, which can lower at most 30% of execution delay compared with baseline algorithms. The optimal ratio of MEC servers and IoT-based UEs is also demonstrated by considering efficient usage of resources and economic costs. Last but not least, our IHRA is suitable for various kinds of applications. Energy assumption will be considered to achieve green city in the near future.

References

[1] J. Huang, C.-C. Xing, and C. Wang, "Simultaneous wireless information and power transfer: Technologies, applications, and research challenges," *IEEE Commun. Mag.*, vol. 55, no. 11, pp. 26–32, Nov. 2017.

[2] Z. Ning *et al.*, "A cooperative quality-aware service access system for social Internet of Vehicles," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2506–2517, Aug. 2018.

[3] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, and F. Giust, "Mobile-edge computing architecture: The role of MEC in the Internet of Things," *IEEE Consum. Electron. Mag.*, vol. 5, no. 4, pp. 84–91, Oct. 2016.

[4] Z. Ning, F. Xia, N. Ullah, X. Kong, and X. Hu, "Vehicular social networks: Enabling smart mobility," *IEEE Commun. Mag.*, vol. 55, no. 5, pp. 16–55, May 2017.

[5] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile edge computing potential in making cities smarter," *IEEE Commun. Mag.*, vol. 55, no. 3, pp. 38–43, Mar. 2017.

[6] L. Yang, J. Cao, H. Cheng, and Y. Ji, "Multi-user computation partitioning for latency sensitive mobile cloud applications," *IEEE Trans. Comput.*, vol. 64, no. 8, pp. 2253–2266, Aug. 2015.

[7] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Barcelona, Spain, Jul. 2016, pp. 1451–1455.

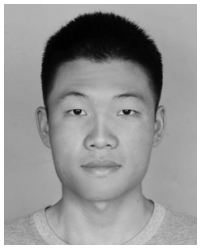
[8] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.

[9] O. Muñoz, A. Pascual-Iserte, and J. Vidal, "Joint allocation of radio and computational resources in wireless application offloading," in *Proc. Future Netw. Mobile Summit*, Lisbon, Portugal, Jul. 2013, pp. 1–10.

[10] O. Muñoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading," *IEEE Trans. Veh. Technol.*, vol. 64, no. 10, pp. 4738–4755, Oct. 2015.
[11] X. Wang, Z. Ning, and L. Wang, "Offloading in Internet of Vehicles: A fog-enabled real-time traffic management system," *IEEE Trans. Ind. Informat.*, to be published, doi: 10.1109/TII.2018.2816590.
[12] W. Labidi, M. Sarkiss, and M. Kamoun, "Joint multi-user resource scheduling and computation offloading in small cell networks," in *Proc. IEEE 11th Int. Conf. Wireless Mobile Comput. Netw. Commun. (WiMob)*, Oct. 2015, pp. 794–801.
[13] M. Kamoun, W. Labidi, and M. Sarkiss, "Joint resource allocation and offloading strategies in cloud enabled cellular networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, London, U.K., Jun. 2015, pp. 5529–5534.
[14] W. Labidi, M. Sarkiss, and M. Kamoun, "Energy-optimal resource scheduling and computation offloading in small cell networks," in *Proc. 22nd Int. Conf. Telecommun. (ICT)*, Sydney, NSW, Australia, Apr. 2015, pp. 313–318.
[15] S. Barbarossa, S. Sardellitti, and P. D. Lorenzo, "Joint allocation of computation and communication resources in multiuser mobile cloud computing," in *Proc. IEEE 14th Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, Darmstadt, Germany, Jun. 2013, pp. 26–30.
[16] W. Hou, Z. Ning, and L. Guo, "Green survivable collaborative edge computing in smart cities," *IEEE Trans. Ind. Informat.*, vol. 14, no. 4, pp. 1594–1605, Apr. 2018.
[17] Y. Zhao, S. Zhou, T. Zhao, and Z. Niu, "Energy-efficient task offloading for multiuser mobile cloud computing," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, Shenzhen, China, Nov. 2015, pp. 1–5.
[18] C. You and K. Huang, "Multiuser resource allocation for mobile-edge computation offloading," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, Washington, DC, USA, Dec. 2016, pp. 1–6.
[19] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.
[20] Y. Mao, J. Zhang, S. H. Song, and K. B. Letaief, "Power-delay tradeoff in multi-user mobile-edge computing systems," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, Washington, DC, USA, Dec. 2016, pp. 1–6.
[21] Z. Ning, X. Wang, X. Kong, and W. Hou, "A social-aware group formation framework for information diffusion in narrowband Internet of Things," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 1527–1538, Jun. 2018.
[22] J. Zhang *et al.*, "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2633–2645, Aug. 2018.
[23] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.
[24] H. Flores *et al.*, "Mobile code offloading: From concept to practice and beyond," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 80–88, Mar. 2015.
[25] X. Wang *et al.*, "A privacy-preserving message forwarding framework for opportunistic Cloud of Things," *IEEE Internet Things J.*, to be published, doi: 10.1109/JIOT.2018.2864782.
[26] C. Wang, F. R. Yu, C. Liang, Q. Chen, and L. Tang, "Joint computation offloading and interference management in wireless cellular networks with mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 66, no. 8, pp. 7432–7445, Aug. 2017.
[27] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 16, no. 8, pp. 4924–4938, Aug. 2017.
[28] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

Zhaolong Ning (M'14) received the M.S. and Ph.D. degrees from Northeastern University, Shenyang, China.

He was a Research Fellow with Kyushu University, Fukuoka, Japan. He is an Associate Professor with the School of Software, Dalian University of Technology, Dalian, China. He has authored or co-authored over 80 scientific papers in international journals and conferences. His current research interests include Internet of Vehicles, edge computing, and mobile computing.



Peiran Dong received the B.S. degree from the Dalian University of Technology, Dalian, China, in 2018, where he is currently pursuing the M.S. degree in software engineering.

His current research interests include mobile edge computing, network computation offloading, and resource management.

Xiangjie Kong (M'13–SM'17) received the Ph.D. degree from Zhejiang University, Hangzhou, China.

He is currently an Associate Professor with the School of Software, Dalian University of Technology, Dalian, China. He has authored or co-authored over 50 scientific papers in international journals and conferences. His current research interests include big traffic data, social computing, and mobile computing.

Feng Xia (M'07–SM'12) received the B.Sc. and Ph.D. degrees from Zhejiang University, Hangzhou, China.

He was a Research Fellow with the Queensland University of Technology, Brisbane, QLD, Australia. He is currently a Full Professor with the School of Software, Dalian University of Technology, Dalian, China. He has authored or co-authored 2 books and over 200 scientific papers in international journals and conferences. His current research interests include computational social science, big data, and mobile social networks.

Dr. Xia has been a Guest Editor of several international journals. He serves as the General Chair, the PC chair, the Workshop Chair, or the Publicity Chair of a number of conferences. He is a Senior Member of the ACM and a member of the AAAS.