Latency Optimization for Heterogeneous Task Offloading in Cooperative MEC Network

Zhiwei Jiang, Yijin Pan and Chenhao Qi
School of Information Science and Engineering, Southeast University, China
National Mobile Communications Research Laboratory, Southeast University, Nanjing, China
Email: {jiangzw, panyj, qch}@seu.edu.cn

2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring) | 979-8-3503-1114-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/VTC2023-Spring57618.2023.10199966

***Abstract*—In this paper, we consider a cooperative mobile edge computing (MEC) network where both the user equipments (UEs) and the MEC server can help with the task computing. Multiple heterogeneous tasks of different sizes for each UE are separately offloaded to the nearby UEs and MEC server's central processing unit and graphics processing unit for processing. Tasks of the same type are transmitted sequentially so that the waiting latency is required for offloading transmission and computation. We aim to minimize the maximum latency of UEs for processing all the tasks while ensuring that all tasks are successfully transmitted and processed. To solve the formulated non-convex problem, an iterative algorithm named directed mutation process based on discrete differential evolution is proposed. Simulation results are presented to verify the performance gain provided by the proposed algorithm.**

***Index Terms*—Latency optimization, mobile edge computing, resource allocation, task offloading.**

I. Introduction

Emerging applications such as augmented reality (AR) and autonomous driving raise the requirements for data processing and graphics rendering capability of the terminal devices [1]. To meet these challenges, mobile edge computing (MEC) has been developed by ETSI to equip high-speed computation unit at the network edge [2], which can provide users with low-latency network service [3].

Due to the potential attractive benefits, various researchers have proposed efficient task offloading approaches in MEC network. In [4], a basic three-node MEC system is proposed to improve the computation capacity and energy efficiency. A joint optimization paradigm for task-driven resource assignment is proposed to optimize the resource allocation strategy [5]. In [6], a fog-computing cloud radio access network is proposed to include three tiers of computational service. The superiority of the network in terms of the delay performance is validated [4]–[7].

Meanwhile, with the rapid development of Internet of things, a fully-connected intelligent world is becoming a reality. If we merely rely on the limited computing resources at the edge of the access network, the tasks generated by numerous of applications may not be finished in time [8]. At the same time, as we enter into the post-Moore era, more and more chips are equipped with powerful computational capacity to obtain higher performance [9]. It is easily happened that some devices are idle when other devices are processing tasks overloaded. In light of the rapid development of central processing units (CPUs) and graphics processing unit (GPUs) equipped by user equipments (UEs), collaborative computing is considered as a promising approach to enhance the MEC service [10]. In cooperative computation, the computing resources within UEs can be shared via device-to-device (D2D) communications. In [11], the integrated optimization problem of D2D communications and MEC is formulated as a mixed integer nonlinear problem, where cooperative approaches have been dedicated to improving the power efficiency. And by enabling bidirectional computation sharing among the users, the cooperative computing design can significantly reduce the system energy consumption [12]. In [13], the penalty of unaccomplished tasks is included into the system cost. Since MEC server's limited computing resources, many tasks need to be queued for processing. In [14], the service queue is modeled as $M/G/n/\infty$ queue model. And in [15], the offloading probability and transmit power of devices are optimized to minimize the average weighted network cost by using $M/M/1$ and $M/M/c$ model.

However, the existing works do not fully consider the heterogeneity of multiple tasks, which requires different chips for processing. In addition, the service queue is usually modeled as a stochastic process, and the queuing delay of each task in sequential transmission and sequential processing are not well addressed. In this context, we consider a cooperative MEC network composed of multiple UEs and a base station (BS), where a MEC server is deployed at the BS. UEs can offload their tasks locally, to the MEC server, or to other UEs via D2D links. All devices are equipped with CPU and GPU to process different types of tasks in different ways. Considering the heterogeneity of tasks, disjoint transmission channels are allocated for them separately. Tasks of the same type are sent in a sequential manner. As a result, the total latency includes not only transmission and processing latency, but also latency of queuing to wait for processing, where the latter are usually ignored in current work. The formulated problem aims to minimize the maximum latency of all devices. Meanwhile, all tasks can be successfully transmitted and processed. An iterative algorithm named directed mutation process based on discrete differential evolution (DMP-DDE) is proposed to solve this mixed integer optimization problem. Compared with local offloading and random offloading, the simulation results are provided to validate the latency performance improvement obtained by the proposed system.

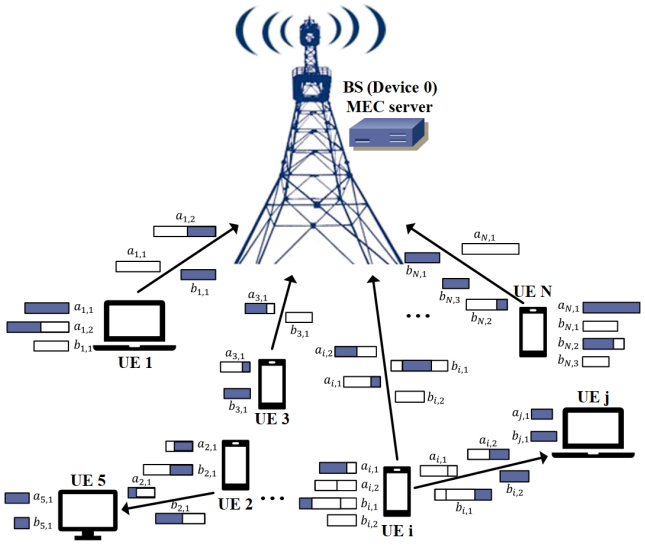


Fig. 1. Network model.

II. Proposed System Model

A. Network Model

As shown in Fig. 1, we consider a cellular network with one BS equipped with a MEC server to serve N UEs. Let $\mathcal{N} \triangleq \{0, 1, 2, \cdots, N\}$ denotes the set of devices, where device 0 is the MEC server. UEs can connect to each other through D2D links. All devices have CPUs and GPUs with different computing capabilities.

Apart from the MEC server, UEs with abundant computation can also help to compute tasks from other UEs. The task model in our work has given fully consideration of the task variation and heterogeneity. We divide UE's tasks into two categories, including CPUs tasks and GPUs tasks.

Given the limited cores of CPUs, it is assumed that the CPU of all UEs process subtasks sequentially. For the CPU tasks generated by UE n, we define vector $\boldsymbol{a}_n$ as

$$\boldsymbol{a}_n \triangleq \left[a_{1,n}, a_{2,n}, \cdots, a_{M_n,n}\right]^{\mathrm{T}} \tag{1}$$

where M_n is the total number of all the indivisible CPU subtasks of UE n. The element $a_{m,n}$ denotes the data size of the CPU subtasks m generated in UE n with $1 \leq m \leq M_n$. Let M denote the maximum number of CPU subtasks among the UEs, i.e., $M \triangleq \max\{m_n, \forall 1 \leq n \leq N\}$. Without loss of generality, it is assumed that the subtasks are sorted in the descending order of data size, i.e., $a_{1,n} > \cdots > a_{M_n,n}$. Therefore, we have the CPU subtask matrix defined as

$$\boldsymbol{A} \triangleq \begin{bmatrix} \boldsymbol{a}_1 & \cdots & \boldsymbol{a}_n & \cdots & \boldsymbol{a}_N \\ \boldsymbol{0}_{M-M_1} & \cdots & \boldsymbol{0}_{M-M_n} & \cdots & \boldsymbol{0}_{M-M_N} \end{bmatrix} \tag{2}$$

where $\boldsymbol{0}_{M-M_n}$ is a column vector with $M - M_n$ zeros.

Due to the different architecture of GPUs, that is, many small cores enabling the parallel acceleration computation, some tasks require GPU processing. For the GPU tasks generated by UE n, we define vector $\boldsymbol{b}_n$ as

$$\boldsymbol{b}_n \triangleq \left[b_{1,n}, b_{2,n}, \cdots, b_{K_n,n}\right]^{\mathrm{T}} \tag{3}$$

where K_n is the total number of all GPU tasks of UE n. The element $b_{k,n}$ denotes the data size of the GPU tasks k generated in UE n with $1 \leq k \leq K_n$. Let K denote the maximum number of GPU tasks among the UEs, i.e., $K \triangleq \max\{k_n, \forall 1 \leq n \leq N\}$. Therefore, we have the GPU task matrix defined as

$$\boldsymbol{B} \triangleq \begin{bmatrix} \boldsymbol{b}_1 & \cdots & \boldsymbol{b}_n & \cdots & \boldsymbol{b}_N \\ \boldsymbol{0}_{K-K_1} & \cdots & \boldsymbol{0}_{K-K_n} & \cdots & \boldsymbol{0}_{K-K_N} \end{bmatrix} \tag{4}$$

where $\boldsymbol{0}_{K-K_n}$ is a column vector with $K - K_n$ zeros.

For UE j's offloading decision of CPU subtasks and GPU tasks to device i, where $0 \leq i \leq N$ and $1 \leq j \leq N$, we define vector $\boldsymbol{x}_{j,i}$ and $\boldsymbol{y}_{j,i}$ as

$$\boldsymbol{x}_{j,i} \triangleq \left[x^j_{i,1}, \cdots, x^j_{i,m}, \cdots, x^j_{i,M_j}\right] \tag{5a}$$

$$\boldsymbol{y}_{j,i} \triangleq \left[y^j_{i,1}, \cdots, y^j_{i,k}, \cdots, y^j_{i,K_j}\right] \tag{5b}$$

where $x^j_{i,m} = 1$ denoting the mth subtask of UE j is offloaded to device i's CPU for processing, and $y^j_{i,k} = 1$ denoting the kth task of UE j is offloaded to device i's GPU for processing. Thus, we have UE j's offloading decision of CPU subtasks matrix defined as $\boldsymbol{X}_j \triangleq \left[\boldsymbol{x}^{\mathrm{T}}_{j,0}, \cdots, \boldsymbol{x}^{\mathrm{T}}_{j,i}, \cdots, \boldsymbol{x}^{\mathrm{T}}_{j,N}\right]^{\mathrm{T}}$. Similarly, we define a matrix $\boldsymbol{Y}_j$ consisting of $N+1$ column vector $\boldsymbol{y}_{j,i}$, which denotes UE j's offloading decision of GPU tasks. Then the offloading decision of all UEs can be denoted as $\mathcal{X} = \{\boldsymbol{X}_1, \cdots, \boldsymbol{X}_N\}$ and $\mathcal{Y} = \{\boldsymbol{Y}_1, \cdots, \boldsymbol{Y}_N\}$.

Therefore, UE j's task offloading results can be represented as

$$\boldsymbol{c}_j = \boldsymbol{X}_j \boldsymbol{a}_j, \quad \boldsymbol{g}_j = \boldsymbol{Y}_j \boldsymbol{b}_j \tag{6}$$

where $\boldsymbol{c}_j$ is a column vector with size $N+1$. Similarly, the GPU task offloading decision can be presented as $\boldsymbol{g}_j$.

In fact, we assume that the devices adopt frequency division duplex to enable transmitting and receiving tasks simultaneously.

B. Tasks Offloading

In the uplink transmission, UEs utilize frequency division multiple access to offload tasks to multiple devices simultaneously. With the given offloading decision $\boldsymbol{X}_j$ and $\boldsymbol{Y}_j$, we let set $\mathbb{C}_j$ denote the set of devices that UE j offloads tasks to. For the device $i \in \mathbb{C}_j$, we have $\|\boldsymbol{x}_{j,i}\|_0 \neq 0$ or $\|\boldsymbol{y}_{j,i}\|_0 \neq 0$. Let l_j denote the number of devices that UE j offloads tasks to, i.e, the cardinality of set $\mathbb{C}_j$. Then, the total uplink bandwidth B is equally divided into l_j"offloading channel" for CPU and GPU task offloading, every channel with a bandwidth of B/l_j.

Meanwhile, a UE may have both CPU subtasks and GPU tasks offloading to the same device, so that disjoint bandwidth is allocated for GPU task offloading and CPU task offloading. To enhance task transmission, we propose a task size proportional allocation method to split the bandwidth for CPU subtasks and GPU tasks. According to (6), the ith elements of $\boldsymbol{c}_j$ and $\boldsymbol{g}_j$, denoted as $c_{i,j}$ and $g_{i,j}$, represent the total CPU subtasks and GPU tasks of UE j offloaded to device i, respectively. Then, with the given offloading decision $\boldsymbol{X}_j$ and $\boldsymbol{Y}_j$, the bandwidth for UE j offloading CPU subtasks to device i is given by

$$B_{j,i}^{c}(\mathcal{X},\mathcal{Y})=\frac{Bc_{i,j}}{l_j\left(c_{i,j}+g_{i,j}\right)}. \tag{7}$$

And the left bandwidth for offloading GPU tasks is

$$B_{j,i}^{g}(\mathcal{X},\mathcal{Y})=\frac{Bg_{i,j}}{l_j\left(c_{i,j}+g_{i,j}\right)}. \tag{8}$$

The channel fading coefficient between UE j and device i is $h_{i,j}$. The channel noise power spectral density is N_0 mW/Hz. Then, the required received power on unit bandwidth for device i with a SNR threshold γ_i is given by $p_{j,i}=\gamma_i N_0/h_{i,j}$. Let P_j denote the transmission power of UE j. Then, the required power for receiving CPU subtasks at device i is given by

$$B_{j,i}^{c}(\mathcal{X},\mathcal{Y})p_{j,i}=B_{j,i}^{c}(\mathcal{X},\mathcal{Y})\frac{\gamma_i N_0}{h_{i,j}}. \tag{9}$$

If $B_{j,i}^{c}(\mathcal{X},\mathcal{Y})p_{j,i}\le P_j$, UE j can offload the CPU subtasks to device i without distortion; otherwise, this channel is infeasible.

Similarly, the required power for receiving GPU tasks at device i is given by

$$B_{j,i}^{g}(\mathcal{X},\mathcal{Y})p_{j,i}=B_{j,i}^{g}(\mathcal{X},\mathcal{Y})\frac{\gamma_i N_0}{h_{i,j}}. \tag{10}$$

The feasibility of GPU task offloading on this channel can be verified in the similar way.

Therefore, the offloading rate from UE j to device i's CPU is

$$C_{j,i}^{c}(\mathcal{X},\mathcal{Y})=B_{j,i}^{c}\log_2\left(1+\frac{P_j}{N_0B_{j,i}^{c}}\right). \tag{11}$$

And the offloading rate from UE j to device i's GPU is

$$C_{j,i}^{g}(\mathcal{X},\mathcal{Y})=B_{j,i}^{g}\log_2\left(1+\frac{P_j}{N_0B_{j,i}^{g}}\right). \tag{12}$$

III. Problem Formulation And Analysis

In this paper, a multiuser collaborative task offloading scheme is investigated. By optimizing the offloading decision, the total latency T of the entire system can be minimized while ensuring that all tasks are completed. Then, the optimization problem can be formulated as

$$\min_{\{\mathcal{X},\mathcal{Y}\}} \quad T \tag{13a}$$

$$\text{s.t.} \quad x_{j,m}^{i},y_{j,k}^{i}\in\{0,1\}, \quad \begin{cases}\forall i\in\mathcal{N}\backslash\{0\}\\ \forall j\in\mathcal{N}\\ \forall m\in\mathcal{M}_i\\ \forall k\in\mathcal{K}_i\end{cases} \tag{13b}$$

$$\sum_{j=1}^{N}x_{j,m}^{i}=1, \quad \begin{cases}\forall m\in\mathcal{M}_i\\ \forall i\in\mathcal{N}\backslash\{0\}\end{cases} \tag{13c}$$

$$\sum_{j=1}^{N}y_{j,k}^{i}=1, \quad \begin{cases}\forall k\in\mathcal{K}_i\\ \forall i\in\mathcal{N}\backslash\{0\}\end{cases} \tag{13d}$$

$$\begin{cases}P_j\ge B_{j,i}^{c}p_{j,i}\\ P_j\ge B_{j,i}^{g}p_{j,i},\end{cases} \quad \begin{cases}\forall j\in\mathcal{N}\backslash\{0\}\\ \forall i\in\mathbb{C}_j\end{cases} \tag{13e}$$

where $\mathcal{M}_i\triangleq\{1,2,\cdots,M_i\}$ and $\mathcal{K}_i\triangleq\{1,2,\cdots,K_i\}$. (13b) guarantees that all tasks cannot to be partitioned during offloading. (13c) and (13d) guarantees that every task only needs and must be processed by a single device. (13e) guarantees that the leveraged offloading channels are feasible. The problem aims to minimize T by optimizing offloading decisions $\{\mathcal{X},\mathcal{Y}\}$.

IV. Cooperative Latency Analysis

Obviously, T is equal to the maximum latency of all devices in the system to complete the task. To simplify the timing analysis, we assume that the CPU and the GPU of the same device can work simultaneously without affecting each other, so that the latency of device i is equal to the greater of the two: the time of its CPU finishing subtask processing and that of GPU's. The former is denoted as T_i^c, and the latter is T_i^g. Due to the different computing capabilities of MEC server and UEs, it is assumed that tasks are processed sequentially. Meanwhile, we assume both the MEC server's CPU and GPU reserve a thread for each UE, so that tasks offloaded from different UEs can be processed in parallel. Thus, we have $T_0^c\triangleq\max\left\{T_{j,0}^c\right\}$ and $T_0^g\triangleq\max\left\{T_{j,0}^g\right\}$, $\forall j\in\mathcal{N}\backslash\{0\}$, where $T_{j,0}^c$ is the latency for MEC server to process all CPU subtasks offloaded from UE j, and $T_{j,0}^g$ is the latency for MEC server to process all GPU tasks offloaded from UE j. Then T can be denoted as

$$T=\max\left\{T_{j,0}^c,T_{j,0}^g,T_j^c,T_j^g\right\},\forall j\in\mathcal{N}\backslash\{0\}. \tag{14}$$

We assume that the size of processing results is usually small enough so that the latency for downloading them can be neglected. Therefore, T_i^c and T_i^g roughly include the task transmission latency and the latency of processing tasks. According to (5a) and (5b), if $x_{i,m}^{j}=1$, the transmission latency of $a_{m,j}$ can be calculated as

$$T_{m,j}^{c,1}=\frac{\sum_{e=1}^{m}x_{i,e}^{j}a_{e,j}}{C_{j,i}^{c}(\mathcal{X},\mathcal{Y})} \tag{15a}$$

considering that tasks of the same category are transmitted sequentially via the same channel. Similarly, if $y_{i,k}^{j}=1$, the transmission latency of $b_{k,j}$ is

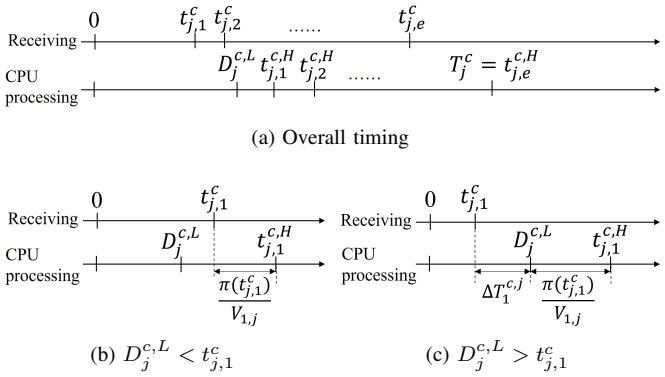
$$T_{k,j}^{g,1}=\frac{\sum_{e=1}^{k}y_{i,e}^{j}b_{e,j}}{C_{j,i}^{g}(\mathcal{X},\mathcal{Y})}. \tag{15b}$$

To simplify the calculation, $V_{1,i}$ and $V_{2,i}$ are used to quantify the computing capabilities of device i's CPU and GPU respectively. And $V_{1,0}$ and $V_{2,0}$ are the computing capabilities of a single thread in MEC server's CPU and GPU.

For each device, the whole process of executing tasks is regarded as a queuing model under the first-input first-output criteria. Due to the difference among the process model of UEs and MEC server, the methods for calculating delays are given respectively.

A. Timing analysis for UE computation

The timing of UE j's CPU subtask execution is shown in Fig. 2. Firstly, UE j processes its local tasks in ascending order of data size sequentially. The processing latency for UE j to complete its local CPU subtasks and GPU tasks are respectively given by



(a) Overall timing

(b) $D_j^{c,L} < t_{j,1}^c$ (c) $D_j^{c,L} > t_{j,1}^c$

Fig. 2. The timing of UE j's CPU subtask execution. (a) denotes the overall timing. (b) and (c) denote the two timings of executing the first offloaded subtasks.

$$D_j^{c,L} = \frac{\sum_{m=1}^{M_j} x_{j,m}^j a_{m,j}}{V_{1,j}}, \tag{16a}$$

$$D_j^{g,L} = \frac{\sum_{k=1}^{K_j} y_{j,k}^j b_{k,j}}{V_{2,j}}. \tag{16b}$$

Then, UE j processes tasks from other UEs according to the arriving order. The arriving time of the first CPU subtask is $t_{j,1}^c = \min\left\{T_{m,i}^{c,1} x_{j,m}^i, \forall x_{j,m}^i \neq 0\right\}$, where $1 \leq i \leq N$. Similarly, we have the following arriving time as $t_{j,2}^c, \cdots, t_{j,e}^c$, which are determined by the ascending order of $\left\{T_{m,i}^{c,1} x_{j,m}^i, \forall x_{j,m}^i \neq 0\right\}$, and $e = \sum_{i \neq j}^{N} \|\boldsymbol{x}_{i,j}\|_0$. We denote the data size of the subtask arrive at $t_{j,k}^c$ as $\pi\left(t_{j,k}^c\right)$. Similarly, we have the arriving time of GPU tasks processed by UE j as $t_{j,1}^g, \cdots, t_{j,f}^g$, which are determined by the ascending order of $\left\{T_{k,i}^{g,1} y_{j,k}^i, \forall y_{j,k}^i \neq 0\right\}$, and $f = \sum_{i \neq j}^{N} \|\boldsymbol{y}_{i,j}\|_0$. The last arriving GPU task is denoted as $\pi\left(t_{j,f}^g\right)$.

To determine the time of finishing the first received CPU subtask, we need to compare $t_{j,1}^c$ with $D_j^{c,L}$. Let $\Delta t_1^{c,j} \triangleq D_j^{c,L} - t_{j,1}^c$. Then the waiting delay for processing $\pi\left(t_{j,1}^c\right)$ is

$$\Delta T_1^{c,j} = \Delta t_1^{c,j} \epsilon\left(\Delta t_1^{c,j}\right) \tag{17}$$

where $\epsilon(t)$ is a sign function. If $t < 0$, we have $\epsilon(t) = 0$; otherwise, we have $\epsilon(t) = 1$. Therefore, the moment when UE j's CPU completes processing $\pi\left(t_{j,1}^c\right)$ is

$$t_{j,1}^{c,H} = t_{j,1}^c + \Delta T_1^{c,j} + \frac{\pi\left(t_{j,1}^c\right)}{V_{1,j}}. \tag{18}$$

Similarly, the kth arriving CPU subtask can be denoted as $\pi\left(t_{j,k}^c\right)$. We can express the difference between $t_{j,k-1}^{c,H}$ and $t_{j,k}^c$ as

$$\Delta t_k^{c,j} = t_{j,k-1}^{c,H} - t_{j,k}^c, \quad k \in \{2, \cdots, e\}. \tag{19}$$

Then the waiting delay for processing $\pi\left(t_{j,k}^c\right)$ is

$$\Delta T_k^{c,j} = \Delta t_k^{c,j} \epsilon\left(\Delta t_k^{c,j}\right). \tag{20}$$

Therefore, the time that UE j's CPU completes processing $\pi\left(t_{j,k}^c\right)$ is

$$t_{j,k}^{c,H} = t_{j,k}^c + \Delta T_k^{c,j} + \frac{\pi\left(t_{j,k}^c\right)}{V_{1,j}}. \tag{21}$$

The time that UE j completes processing all CPU subtasks is

$$T_j^c = t_{j,e}^{c,H} = t_{j,e}^c + \Delta T_e^{c,j} + \frac{\pi\left(t_{j,e}^c\right)}{V_{1,j}}. \tag{22}$$

Similarly, T_j^g can be calculated by

$$T_j^g = t_{j,f}^{g,H} = t_{j,f}^g + \Delta T_f^{g,j} + \frac{\pi\left(t_{j,f}^g\right)}{V_{2,j}} \tag{23}$$

where $\Delta T_f^{g,j}$ denoting the waiting delay for processing $\pi\left(t_{j,f}^g\right)$ can be computed by (24). In fact, we have $t_{j,f}^g = \max\left\{T_{k,j}^{g,1} y_{j,k}^i, \forall y_{j,k}^i \neq 0\right\}$, which denotes the time that $\pi\left(t_{j,f}^g\right)$ arrives. We also have

$$\Delta T_f^{g,j} = \Delta t_f^{g,j} \epsilon\left(\Delta t_f^{g,j}\right) \tag{24}$$

where $\Delta t_f^{g,j} = t_{j,f-1}^{g,H} - t_{j,f}^g$. $t_{j,f-1}^{g,H}$ denotes the moment when UE j completes processing $(f-1)$th GPU task. When $f = 1$, $\Delta t_{1,1}^{g,j}$ is calculated as

$$\Delta t_{1,1}^{g,j} = D_j^{g,L} - t_{j,1}^g. \tag{25}$$

B. Timing analysis at MEC server

MEC server processes tasks from the same UE according to the arriving order. We have the arriving time of CPU subtasks from UE j as $T_{1,j}^{c,1}, \cdots, T_{s,j}^{c,1}$, where $s = \|\boldsymbol{x}_{j,0}\|_0$. Denote the data size of the subtask arrive at time $T_{1,j}^{c,1}$ as $a_{1,j}^0$. Similarly, we have the arriving time of GPU tasks from UE j as $T_{1,j}^{g,1}, \cdots, T_{u,j}^{g,1}$, where $u = \|\boldsymbol{y}_{j,0}\|_0$.

Therefore, the time when MEC server's CPU completes processing $a_{1,j}^0$ is

$$t_{j,0}^{c,1} = T_{1,j}^{c,1} + \frac{a_{1,j}^0}{V_{1,0}}. \tag{26}$$

Similar to the aforementioned calculation process, $T_{j,0}^c$ can be calculated as

$$T_{j,0}^c = T_{j,0}^{c,s} = T_{s,j}^{c,1} + \frac{a_{s,j}^0}{V_{1,0}} + \Delta T_{1,s-1}^{c,0}. \tag{27}$$

Note that $\Delta T_{1,s-1}^{c,0}$ denoting the waiting delay for processing $a_{s,j}^0$ is calculated by

$$\Delta T_{1,s-1}^{c,0} = \Delta t_{1,s-1}^{c,0} \epsilon\left(\Delta t_{1,s-1}^{c,0}\right) \tag{28}$$

where $\Delta t_{1,s-1}^{c,0} = T_{j,0}^{c,s-1} - T_{s,j}^{c,1}$. $T_{j,0}^{c,s-1}$ denotes the time that MEC server's CPU completes processing $(s-1)$th subtasks from UE j.

Similarly, $T_{j,0}^g$ can be calculated by

$$T_{j,0}^g = T_{j,0}^{g,u} = T_{u,j}^{g,1} + \frac{b_{u,j}^0}{V_{2,0}} + \Delta T_{1,u-1}^{g,0}. \tag{29}$$

Algorithm 1 DMP-DDE Algorithm

Initialize P, ϵ and $S_{\max}$; Set the iteration number $s = 0$.
Initialize $\mathcal{W}_{(1,0)}$ according to (30), and initialize $\mathcal{W}_{(p,0)}, \forall p \neq 1$ randomly with (13e) satisfied.
while $s \leq S_{\max}$ **do**
  for $p = 1$ to P **do**
    Calculate the objective $T_{(p,s)}$ obtained by $\mathcal{W}_{(p,s)}$.
    Generate $\mathcal{V}_{(p,s)}$ via (31) and repeat the process to ensure that (13c) and (13d) are satisfied.
    Generate $\mathcal{U}_{(p,s)}$ through binomial crossover as:
    if $\mathrm{rand_{p,s}}[0,1] \leq \epsilon$ or $p = p_{\mathrm{rand}}$ **then**
      Set $\mathcal{U}_{(p,s)} = \mathcal{V}_{(p,s)}$;
    else
      Set $\mathcal{U}_{(p,s)} = \mathcal{W}_{(p,s)}$.
    end if
    if $\mathcal{U}_{(p,s)}$ does not satisfy (13e) **then**
      Set $\mathcal{U}_{(p,s)} = \mathcal{W}_{(p,s)}$.
    end if
    Calculate the objective $\hat{T}_{(p,s)}$ obtained by $\mathcal{U}_{(p,s)}$.
    if $\hat{T}_{(p,s)} \leq T_{(p,s)}$ **then**
      Set $\mathcal{W}_{(p,s)} = \mathcal{U}_{(p,s)}$.
    end if
  end for
  $T_{(s)}^{\min} = \min\{T_{(p,s)}, \forall 1 \leq p \leq P\}$.
  $s \leftarrow s + 1$.
end while

V. Task Offloading Scheme Based on DDE

It can be verified that the formulated problem is non-convex and hard to solve. To be specific, the allocated bandwidth and processioning latency for task computation are highly dependent on the offloading decision $\mathcal{X}$ and $\mathcal{Y}$. Based on the DDE algorithm [16], we develop the DMP-DDE algorithm to solve (13).

In DMP-DDE algorithm, the set of candidate offloading solutions are represented as a population with size P. Then, the p-th candidate offloading solution at s-th iteration is denoted as $\mathcal{W}_{(p,s)} = \{\mathcal{X}_{(p,s)}, \mathcal{Y}_{(p,s)}\}$. Note that a proper initialization method can help improving the convergence of the algorithm. Thus, we initialize the first candidate solution $\mathcal{W}_{(1,0)} = \{\mathcal{X}_{(1,0)}, \mathcal{Y}_{(1,0)}\}$ as local computation for all UEs. To be more specific, we have

$$\mathcal{X}_{(1,0)} = \{\boldsymbol{X}_1^0, \cdots, \boldsymbol{X}_N^0\}, \quad \mathcal{Y}_{(1,0)} = \{\boldsymbol{Y}_1^0, \cdots, \boldsymbol{Y}_N^0\}. \quad (30)$$

The $(i+1)$th columns of both $\boldsymbol{X}_i^0$ and $\boldsymbol{Y}_i^0$ are all ones, i.e., $\boldsymbol{x}_{i,i} = \mathbf{1}_{1\times(N+1)}, \boldsymbol{y}_{i,i} = \mathbf{1}_{1\times(N+1)}$, and other columns are zeros. Then, $\boldsymbol{X}_i^0$ and $\boldsymbol{Y}_i^0$ represent that all GPU tasks and CPU tasks are processed locally. In this way, the local computation solution is included in the set of candidate offloading solutions.

We predefine a maximum number of iterations $S_{\max}$, and the following processes are run iteratively until $S_{\max}$ iterations is finished. In the mutation process of the algorithm, the difference between two randomly selected candidate solutions $\mathcal{W}_{(i,s)}$ and $\mathcal{W}_{(j,s)}$ are added to the first candidate solution $\mathcal{W}_{(1,s)}$ to generate a mutation solution. To be specific, the mutation solution of s-th iteration is

$$\mathcal{V}_{(p,s)} = \mathrm{mod}_2(\mathcal{W}_{(1,s)} + (\mathcal{W}_{(i,s)} - \mathcal{W}_{(j,s)})) \quad (31)$$

where $\mathrm{mod}_2(\cdot)$ is introduced to ensure that the solutions satisfy (13b).

Then a crossover operator ϵ is introduced to combine the mutation solution $\mathcal{V}_{(p,s)}$ with the candidate solution $\mathcal{W}_{(p,s)}$ so as to generate a trial solution $\mathcal{U}_{(p,s)}$. Thereafter, a selection process is applied to compare the objective function value of both candidate solutions and trial solutions to determine who can survive in the next iteration.

The above procedure is summarized in **Algorithm 1** named as the DMP-DDE algorithm. Practically, all UEs send signals to the BS, including the information of tasks to be offloaded. The proposed algorithm is conducted in the central MEC server to obtain the offloading decisions, which are then sent to all UEs.

VI. Simulation Results

In this section, MATLAB is used as a simulation tool to show the performance improvement obtained by the proposed system and the DMP-DDE algorithm. We assume that the computing capabilities of UE j's CPU is $V_{1,j} \in [1, 400]$ KB/s, and that of GPU is $V_{2,j} \in [10000, 200000]$ KB/s. The path loss adopts the ITU-1411 model. Other simulation parameters are presented in Table I.

TABLE I: SIMULATION PARAMETERS

Parameters	Value
Number of UEs N	4∼49
Data size of CPU subtasks $a_{m_n,n}$	(0,2] KB
Maximum number of subtasks generated by a UE M	5∼15
Data size of GPU tasks $b_{k_n,n}$	(0,2] MB
Maximum number of tasks generated by a UE K	5∼15
Distance between devices $D_{i,j}$	[1,50] m
Bandwidth B	20 MHz
Noise power density N_0	-174 dBm/Hz
SNR threshold of each UE γ_j	-85∼(-95) dBm
SNR threshold of BS γ_0	-120 dBm
Transmit power of each UE P_j	0∼20 dBm
Transmit power of BS P_0	40 dBm
Computing capability of each thread of MEC server's CPU $V_{1,0}$	450 KB/s
Computing capability of each thread of MEC server's GPU $V_{2,0}$	250000 KB/s

Fig. 3 shows the convergence of the proposed algorithm. In Fig. 3, we fix N firstly, then generate $a_{m_n,n}, M, b_{k_n,n}, K, D_{i,j}, \gamma_j$ and P_j randomly to obtain the corresponding scene, and then use the proposed algorithm to optimize the scene for 10 times to test the performance of the algorithm. It is shown that the total latency obtained by the proposed algorithm decreases with increasing iterations and the effectiveness has been verified. In addition, by comparing the best and the average of 10 times of optimization, Fig. 3 shows that the performance gap between the average convergence results and the best of ten are very small for all considered cases. This also shows that the performance of the proposed algorithm is quite stable and may avoid the local optimum.

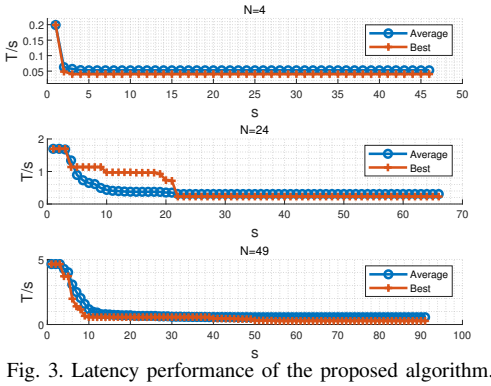


Fig. 3. Latency performance of the proposed algorithm.

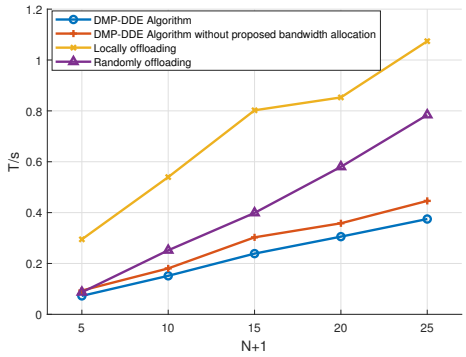


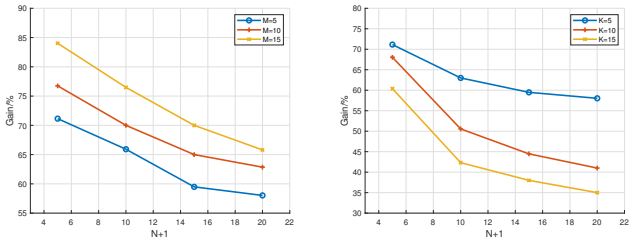
Fig. 4. The latency of MEC system under different offloading decisions.

Fig. 4 shows the latency of MEC system under different offloading decisions. In Fig. 4, the impact of the proposed bandwidth allocation method is also investigated, and the "Orange" line marked with "+" represents the DMP-DDE algorithm with equally allocated bandwidth. It can be verified that the offloading decision obtained by the proposed algorithm is significantly better than locally offloading and randomly offloading. In addition, the task size proportional allocation method proposed to split the bandwidth can also reduce the system latency.

Fig. 5 shows the performance of the algorithm in general scenarios. It shows that the gain decreases as the number of devices increases, where the gain is the ratio of the optimization amount to the latency of all tasks processed locally. It can also be concluded that tasks with larger size of data, such as GPU tasks, are better for local computing.

VII. Conclusion

In this work, we have investigated the latency optimization for heterogeneous task offloading in the cooperative MEC network. Tasks are divided into two categories, and they can be offloaded locally, offloaded to the MEC server, or offloaded to other UEs. We have considered the sequential transmission of multiple tasks and their sequential and parallel processing. We have analyzed the transmission and computation timing in combination with the characteristics of CPUs and GPUs. To solve this non-convex problem, we have proposed the DMP-DDE algorithm. It is shown that the cooperative MEC network and the DMP-DDE algorithm can significantly improve the latency performance, and the task size proportional allocation method proposed to split the bandwidth can also decrease the latency.



(a) K=5 (b) M=5

Fig. 5. The performance of the algorithm in general scenarios.

Acknowledgment

This work is supported in part by National Natural Science Foundation of China (NSFC) under Grants 62071116 and 62001107.

References

[1] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet of Things J.*, vol. 3, no. 6, pp. 854–864, Dec. 2016.

[2] E. T. S. I. (ETSI), "Mobile-edge computing—Introductory technical white paper," 2014.

[3] E. Baccarelli, M. Scarpiniti, P. G. V. Naranjo, and L. Vaca-Cardenas, "Fog of social IoT: When the fog becomes social," *IEEE Network*, vol. 32, no. 4, pp. 68–80, Aug. 2018.

[4] X. Cao, F. Wang, J. Xu, R. Zhang, and S. Cui, "Joint computation and communication cooperation for energy-efficient mobile edge computing," *IEEE Internet of Things J.*, vol. 6, no. 3, pp. 4188–4200, Jun. 2019.

[5] L. Wan, L. Sun, X. Kong, Y. Yuan, K. Sun, and F. Xia, "Task-driven resource assignment in mobile edge computing exploiting evolutionary computation," *IEEE Wireless Commun.*, vol. 26, no. 6, pp. 94–101, Dec. 2019.

[6] Y. Pan, H. Jiang, H. Zhu, and J. Wang, "Latency minimization for task offloading in hierarchical fog-computing C-RAN networks," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, Jun. 2020, pp. 1–6.

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

[8] M. Liu and Y. Liu, "Price-based distributed offloading for mobile-edge computing with computation capacity constraints," *IEEE Wireless Commun. Lett.*, vol. 7, no. 3, pp. 420–423, Jun. 2018.

[9] N. S. Kim, D. Chen, J. Xiong, and W.-m. W. Hwu, "Heterogeneous computing meets near-memory acceleration and high-level synthesis in the post-moore era," *IEEE Micro*, vol. 37, no. 4, pp. 10–18, Aug. 2017.

[10] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 54–61, Apr. 2017.

[11] Y. He, J. Ren, G. Yu, and Y. Cai, "D2D communications meet mobile edge computing for enhanced computation capacity in cellular networks," *IEEE Trans. Wireless Commun.*, vol. 18, no. 3, pp. 1750–1763, Mar. 2019.

[12] Q. Lin, F. Wang, and J. Xu, "Optimal task offloading scheduling for energy efficient D2D cooperative computing," *IEEE Commun. Lett.*, vol. 23, no. 10, pp. 1816–1820, Oct. 2019.

[13] Y. Pan, C. Pan, K. Wang, H. Zhu, and J. Wang, "Cost minimization for cooperative computation framework in MEC networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 6, pp. 3670–3684, Jun. 2021.

[14] S. Guo, D. Wu, H. Zhang, and D. Yuan, "Queueing network model and average delay analysis for mobile edge computing," *Int. Conf. Comput., Netw. Commun.*, pp. 172–176, 2018.

[15] L. Liu, Z. Chang, X. Guo, S. Mao, and T. Ristaniemi, "Multiobjective optimization for computation offloading in fog computing," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 283–294, 2018.

[16] Q.-K. Pan, L. Wang, and B. Qian, "A novel differential evolution algorithm for bi-criteria no-wait flow shop scheduling problems," *Comput. Oper. Res.*, vol. 36, no. 8, pp. 2498–2511, Oct. 2009.