Delay Minimization Offloading for Interdependent Tasks in Energy-Aware Cooperative MEC Networks

Yao Zhu, Yulin Hu*, and Anke Schmeink
ISEK Research Group, RWTH Aachen University, 52062 Aachen, Germany.
Email: $zhu|hu|schmeink$@ti.rwth-aachen.de

***Abstract*—The partial offloading technologies in the cooperative mobile edge computing (MEC) networks are considered as promising solutions to enable the emerging latency-sensitive and compute-intensive applications. In this paper, we characterize the performance model of an energy-aware MEC networks with multiple servers cooperatively computing a set of interdependent tasks. To minimize the total delay of the whole process of the set of tasks, an optimal offloading design is provided under given energy constraints. In particular, we provide an optimal solution to the offloading problem, which makes a 3-dimensional decision (matrix) representing at which time instant to offload which task to which server. Via simulation, we investigate the performance of the proposed design for the tasks with different interdependency structures. In particular, the impacts of the number of MEC servers, CPU frequencies and in the proposed algorithm on the system performance are studied. In addition, the tradeoff between the delay performance and computation complexity is addressed.**

Keywords*—Edge computing, delay minimization, interdependent tasks, offloading

I. Introduction

Recent advancements in Internet-of-Things (IoT) have enabled many computation-intensive and latency-critical applications [1], such as image/environment recognition in augmented reality (AR), autonomous driving, unmanned aerial vehicle and industrial automation. However, IoT devices are usually resource-constrained due to their small physical size, i.e., lack of computation power and energy supply [2] to finish these computation-intensive tasks within a critical time deadline. The mobile cloud computing (MCC) has emerged as a potential solution to enhance the computing capability for IoT applications by offloading IoT devices' computation tasks to the centralized cloud server with both sufficient computing and storage capabilities [3]. However, as the centralized cloud in the MCC servers are far apart from the users in the perspective of both geography and logic, offloading to the MCC server costs considerable latency, i.e., being not able to guarantee the latency requirements of latency-critical IoT applications.

Recently, mobile edge computing (MEC) [4], [5] has been considered as a promising solution addressing both the above computation and latency issues. In particular, instead of offloading tasks to the centralized servers, the IoT devices in a MEC network are able to offload their tasks to the nearby MEC servers, e.g., small-cellular base stations and WiFi access points, deployed at the edge of networks, which significantly reduces the transmission delay in comparison to MCC. On the other hand, unlike the mega data center in the MCC, the servers in MEC are usually with relatively limited computational capability. In particular, when having multiple compute-intensive tasks, a single MEC server is unlikely able to finish them timely by its own computational capability. Observing this, cooperative offloading mechanisms [6]–[8], [10] have been proposed to balance the computing tasks among MEC servers, which enhance the computation capability and reduce the latency for the applications. For instance, the work in [6] proposed a three-node cooperation scheme in a computation and communication scenario by jointly optimizing the task partition and power allocation in a time-slotted fashion. The authors in [8] exploited a peer-to-peer cooperative computing to optimize the workloads on the servers. Furthermore, they developed a Lyapunov optimization based online algorithm and an autonomous peer offloading scheme. The work in [7] provided a resource allocation scheme among the virtual machines to minimize the overall power consumption and formulated it as a mixed integer linear problem. To tackle the new challenges in ultra-reliable low-latency communications [9], the authors in [10] designed a framework to partition a task into subtasks and offload them to multiple MEC servers by taking both latency and reliability into account. It is evident that the above data-partial offloading algorithms reduce the latency of the IoT applications and improve the energy efficiency of the network. However, applying such data-partial offloading algorithms requires the condition that data in a task set is bitewise independent and can be arbitrarily divided into different tasks, which is not true in most practical systems due to the software and hardware constraints [4]. For example, in a video navigation application [12], the final results depend on the results of the graphics, video processing and face detection. In other words, the interdependency among the tasks are needed to be considered in MEC offloading designs.

Nevertheless, the interdependency of tasks has been investigated in [11] in MCC scenarios, following which a heuristic offloading scheme was proposed to minimize the completion delay. Moreover, considering also the interdependency of tasks, the authors in [13] studied the energy-efficient dynamic offloading in MCC by jointly optimizing the offloading decisions, power allocation and frequency control. However, these results are conducted under a MCC scenario, while an optimal design addressing the offloading with cooperative servers in a MEC network is missing. Note that a MEC network has the following difference in comparison to a MCC network: *i.* Each server in the MEC network has its own power consumption limit and computing frequency limits. *ii.* The capabilities for supporting parallel task execution are different, i.e., an extremely larger number of tasks can be computed in parallel in the MCC server, which is definitely not true for a MEC server. *iii.* The cost of the communication between the MEC servers is significantly larger than the cost of exchanging data between virtual machines in the centralized data center of MCC, which makes in the MEC network the communication delay contributes to the total delay in a

*Y. Hu is the corresponding author.

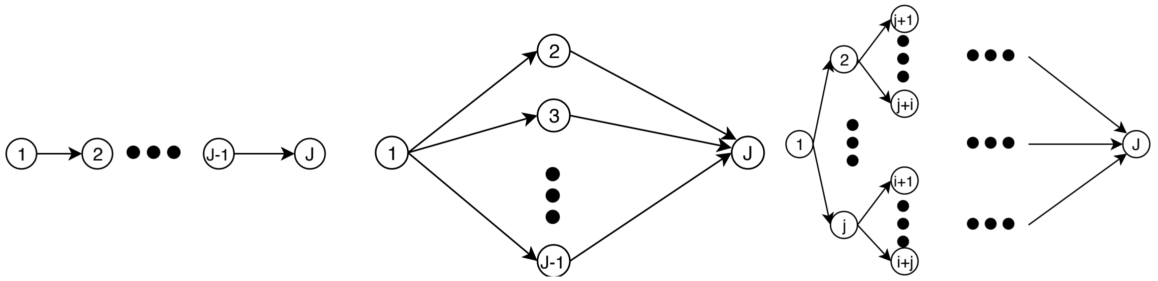

Fig. 1. Typical structures of DAGs.

significantly different way than the MCC network. Hence, it is essential to characterize the performance of a MEC network supporting interdependent tasks and to provide an optimal offloading design for such network to minimize the delay of applications.

In this paper, we consider a MEC network with multiple servers supporting a compute-intensive application with interdependent tasks. We aim at minimizing the delay of the application by making the optimal offloading decisions, i.e., deciding which task should be offloaded to which server at what time, while taking into account the total energy constraint and the individual energy constraint at each server.

II. System Model

We consider a cooperative MEC network with $K+1$ nodes, including an IoT device and K MEC servers. The IoT device collects local information, which is a set of compute-intensive data tasks, and requests the processing/computing results for the further actions. The device is assumed to be unable to compute these tasks locally while all the tasks need to be offloaded to the K MEC servers and computed there. Therefore, the set of nodes involved in the offloading process, including the device and K servers, is given by $\mathcal{K} \triangleq \{1,...,K+1\}$ with $K+1$ elements, where the first element represents the device and the rest elements indicate the K servers.

We call by a service period the total time length for offloading, computing all the tasks and transmitting the result back to the device. In each service period, a set of J tasks denoted by $\mathcal{J} \triangleq \{1,...,J\}$, are generated from the device, which are assumed to be interdependent, i.e., the computed result of one task is required to be known before processing another task. In addition, each task is assumed as the smallest independent computing data unit, i.e., can not be further divided and computed separately. This interdependency relationship among tasks can be described as a directed acyclic graph (DAG) [11]. We denote by $\mathcal{G} = (\mathcal{J}, \mathcal{E})$ the DAG where $\mathcal{E}$ is the set of edges and given by

$$\mathcal{E} = \begin{bmatrix} 0 & e_{12} & e_{13} & \dots & e_{1J} \\ 0 & 0 & e_{23} & \dots & e_{2J} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & e_{(J-1)J} \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}, \tag{1}$$

in which, $e_{ij} \in \{0,1\}$ where $e_{ij}=1$ ($e_{ij}=0$) indicates the execution of the j-th task (not) requires the results of the i-th task.When $e_{ij}=1$, the i-th task is called by the predecessor of j-th task and in the other way round, the j-th task is called by the successor of the i-th task. Note that in a DAG, the interdependency of the tasks follows an acyclic positive direction. Hence, $e_{ij}=0$ holds if $i \geq j$.

The tasks are different and may require different computing workloads. We denote by c_i the required workloads for computing the i-th task and by d_i the data size of the computed results. In the k-th server, we assume that there are n_k homogeneous virtual machines (VM) available for the incoming tasks, while each VM has computation power f_k. We assume that all the MEC servers are connected with a local area network and able to communicate to each other through wired link, while the offloading from the device to MEC servers and the downloading of the computed result of the set of tasks from a server to the device are via wireless link. Note that the offloading node set includes $K+1$ nodes, where the first element is the device. Following this node index structure, we denote by P_k the transmit power of the k-th node, i.e., P_1 is the transmit power of the device. For transmission links of task offloading and result downloading, the bandwidth is the same as B. For the wireless transmission from the l-th node to the k-th node, where $l,k \in \mathcal{K}$, we denote the channel gain of the link by $H_{l,k}$ and the corresponding noise power by $\sigma^2_{l,k}$. Specially, $H_{k,k}$ indicates the channel gain to it self, i.e., $H_{k,k}=\infty$.

Note that the interdependency among tasks is a key concern in our work. In particular, different interdependency relationships correspond to different DAG structures. As examples, we provide in Fig 1 three typical structures of DAGs [11], which are considered in our simulations in Section V. These DAG structures are *(a) Serial structure*: Each task has one and only one successor beside the last task while each task has also only one predecessor except the first task, i.e., $e_{j(j+1)}=1, \forall j \neq J$. *(b) Parallel structure*: The first task is the predecessor of all other tasks except the last task which is the successor of all other tasks except the first task, i.e., $e_{1(j+1)}=1, \forall j \neq J-1$ and $e_{(j+1)1}, \forall j \neq J-1$. *(c) General structure*: It can be decomposed into multiple sub-DAGs with either serial structure or parallel structure. Without loss of generality, we merge the task offloading and result downloading processes between the device and servers into the application by considering the local data as the output of first task and the results of the application as the input of the last task, respectively. Therefore, the 1-st task and J-th task are considered as "virtual" tasks with empty workloads $c_1=c_J=0$. As a result, the CDGs satisfy following properties [12]: *i*) Each task except the first task should have at least one predecessor; *ii*) Each task except the last task should have at least one successor; *iii*) all of the tasks should have at least one direct or indirect path from the first task; *iv*) each task has at least one direct/indirect path to the last task.

III. Performance Characterization

In this section, we study the system behavior of the considered MEC network supporting interdependent tasks offloaded from the device. We first model the decision matrix of the offloading process, following which the transmission delay, the computation delay, energy consumption are characterized.

A. Offloading Decision Matrix Modeling

Existing works in [13] and [14] for the offloading in MCC network propose to model the decision of the offloading by a classical 2-dimensional binary decision matrix, based on which the offloading problem is NP-hard and cannot be solved optimally. Observing this, for considered MEC network we propose to model the offloading decision as a 3-D time-decision matrix. We will show in Section IV that based this 3-D decision model, the offloading problem can be optimally solved. We denote this 3-D decision matrix by $\mathbf{X}$, in which the element $x_{j,k,t} \in \{0,1\}$ indicates whether the j-th task is completed by the k-th node at the end of time slot $t \in \{0,..,T_{J,\max}\}$, where $T_{J,\max}$ is the completion delay constraint of the application.

Based on $\mathbf{X}$, the computation indicator matrix can be obtained, which indicates (based on the offloading decision) each task will be computed at which node. We denote the computation indicator matrix by $\mathbf{A}$ with elements $a_{j,k}$

$$a_{j,k} = \sum\nolimits_t x_{j,k,t}, \tag{2}$$

where $j \in \mathcal{J}$ and $k \in \mathcal{K}$. In fact, $a_{j,k} = 1$ indicates the j-th task will be computed in the k-th node. To avoid the waste of computation resource, we assume that each task is computed only in one server and only computed once. We formulate this execution constraint as follows

$$\sum\nolimits_t \sum\nolimits_k x_{j,k,t} = \sum\nolimits_k a_{j,k} = 1, \tag{3}$$

Denote by t_j the completion time duration for the j-th task. Hence, we have

$$t_j = \sum\nolimits_k \sum\nolimits_t t x_{j,k,t}. \tag{4}$$

B. Communication throughput and delay

The communications occur in our system either between the device and the servers via wireless channels, or between servers themselves via wired link. We denote $R_{l,k}$ the rate of the transmission from the l-th and the k-th node.

At the beginning of each service period, the tasks are offloaded from the device to a node k, the transmission rate is given by

$$R_{1,k} = B\log_2\Big(1 + \frac{P_1 H_{1,k}}{\sigma_k^2}\Big), \tag{5}$$

where $k \in \{2, \cdots K+1\}$, i.e., the k-th node is a server. Similarly, the transmission rate for the results downloading from the l-th node, i.e., $l \in \{2, \cdots K+1\}$, to the device as

$$R_{l,1} = B\log_2\Big(1 + \frac{P_l H_{l,1}}{\sigma_1^2}\Big). \tag{6}$$

In addition, the transmission rate between nodes l and k is denoted by $R_{l,k}$ where $l,k \in \{2, \cdots K+1\}$ and $l \neq k$. In other words, $R_{l,k}$ represents the transmission rate between two MEC server via the wired link. We assume the transmission rates between different servers are the same. Specially, we define $R_{l,k} = \infty$, for $l = k$, to maintain the consistence of the notations.

When the i-th task is the predecessor of the j-th task and they are computed at different servers, transmission occurs. In particular, node l sends the result of the i-th task to node $k > 1$ and $k \neq l$ which will compute the j-th task. The delay of the above transmission is given by

$$t^r_{i,l,j,k} = e_{i,j} a_{j,k} a_{i,l} \frac{d_i}{R_{l,k}}. \tag{7}$$

Note that the above $t^r_{i,l,j,k}$ actually has a quadratic expression due to the fact that it contains the multiplication between $a_{j,k}$ and $a_{i,l}$, which results a relatively higher complexity in the offloading design than a linear objective function. In fact, the multiplication of two binary variables is mathematically equal to an **AND** operation. Hence, the expression (7) can be linearized by introducing variables $\beta_{i,l,j,k} \in \{0,1\}$, $t^r_{i,l,j,k}$, reads as

$$t^r_{i,l,j,k} = e_{i,j} \beta_{i,l,j,k} \frac{d_i}{R_{l,k}}, \tag{8}$$

where $\beta_{i,l,j,k}$ satisfies

$$\beta_{i,l,j,k} \geq a_{j,k}, \tag{9a}$$
$$\beta_{i,l,j,k} \geq a_{i,l}, \tag{9b}$$
$$\beta_{i,l,j,k} \geq 0, \tag{9c}$$
$$\beta_{i,l,j,k} \leq a_{j,k} - (1 - a_{i,l}). \tag{9d}$$

C. Computation delay

Note that the server (k-th node) is unable to start computing the j-th task unless all the required results from depended previous tasks are transmitted to it. Hence, denote by T_j the ready time (point) [13] for starting computing the j-th task at the k-th node, i.e., at the time point T_j after the beginning of the service period, j-th task is started to be computed. In other words, T_j implies the maximal time for all results of the predecessors for the j-task uploaded to k-th server. In this paper, we neglect the I/O processing time between VMs in the same server, as it is relatively small compared to the computation time and transmission time. Therefore, the ready time for the j-th task is given by

$$T_j = \max_{i<j} e_{i,j}(t_i + \sum\nolimits_k \sum\nolimits_l t^r_{i,l,j,k}) \tag{10}$$

Therefore, the completion time of the j-th server can not be earlier than the summation of computation time and the ready time, i.e.,

$$t_j \geq T_j + t^c_j. \tag{11}$$

Note that each server follows a ”first come, first served” principle, i.e., new arrived task in an occupied server must wait in the queue till all previous buffered tasks are computed. We assume all VMs in each server share a global queue to maintain the global load balancing in the physical CPU cores [16]. In particular, at node $k > 1$, the total executed tasks in any computation period t^c_j should not exceed the maximal available number of VMs n_k. The set of constraints are expressed by

$$\sum_j \sum_{t=s}^{\bar{t}} x_{j,k,t} \leq n_k, \tag{12}$$

for different starting time point $s \in \{1, 1+\Delta_s, 1+2\Delta_s \cdots, T_{J,\max}\}$. In addition, $\bar{t} = \min\{s + t_j^c - 1, T_{J,\max}\}$ with $T_{J,\max}$ being the delay tolerance of the application. The constraints of (12) actually describe a set of windows with the size of $\bar{t}$ and J, while in each window the sum of element values should be limited by n_k. Δ_s is actually the time distance between two adjoin windows, represents the resolution for the checking. The benefit of such check window constraints is to monitor the current execution stats of the k-th server without introducing the iteration of decision element $x_{j,k,t}$.

After the ready time, a task is computed. The computation time of the j-th task in the k-th node ($k \in \{2, \cdots K+1\}$) is given by

$$t_j^c = \sum_k a_{j,k} \frac{c_j}{f_k}, \tag{13}$$

D. Energy consumption

The energy consumption at each node contains two parts with respects to transmission and computation. In particular, the transmission energy at the k-th node is consumed for transmitting the results of the predecessors to the corresponding server of the successors

$$E_k^r = \sum\nolimits_j \sum\nolimits_{i<j} \sum\nolimits_l t_{il,jk}^r P_k. \tag{14}$$

On the other hand, the energy consumption for the computation is proportional to the workload c_i and the square of CPU-cycles frequency f_k^2. For the k-th node, the energy consumption is given by [17]:

$$E_k^c = \sum\nolimits_j a_{jk} \kappa c_j f_k^2, \tag{15}$$

where κ is a constant related to the hardware architecture.

Hence, the energy consumption of for k-th server is

$$E_k = E_k^r + E_k^c = \sum_j a_{jk} \left(\kappa c_j f_k^2 + \sum_{i>j} \sum_l e_{ji} \frac{d_j a_{il}}{R_{kl}} P_k \right). \tag{16}$$

The total energy consumption is then the summation of energy consumption of all servers, given by

$$E_O = \sum_k E_k = \sum_j \sum_k a_{jk} \left(\kappa c_j f_k^2 + \sum_{i \leq j} \sum_l e_{ij} \frac{d_i a_{il}}{R_{lk}} P_k \right). \tag{17}$$

IV. Optimal Offloading

In this paper, we aim at an optimal offloading strategy following which the optimal offloading decision (matrix) is made such that the completion delay (equivalently the length of the service period) of the set of tasks is minimized while fulfilling the given energy constraints. This delay minimization problem is formulated in (18), where $\bar{t} = \min\{s + t_j^c - 1, T_{J,\max}\}$. In particular, (18b) is the energy constraints for the k-th node, which implies the available resource of the node may differ from each other not only in terms of the CPU-cycles frequency and VMs but also in terms of the energy. (18c) is the overall energy constraint in the whole service period. Constraint (18d) guarantees that the whole service period should satisfy the delay tolerance. To ensure the uploading of the local data from the device to the servers and the downloading of the results from the servers to the device, the decisions of the first and last task is formulated as constraint (18h). As discussed in previous sections, the constraint of completion time for current j-th task and the constraint of available VMs in the k-th server are formulated as (18f) and (18g), respectively.

$$\begin{aligned}
\underset{\mathbf{X}}{\text{minimize}} \quad & t_J && \text{(18a)}\\
\text{subject to} \quad & E_k \leq E_{k,\max},\ \forall k \in \mathcal{K}, && \text{(18b)}\\
& E_O \leq E_{O,\max}, && \text{(18c)}\\
& t_j \leq T_{J,\max}, && \text{(18d)}\\
& T_j + t_j^c \leq t_j,\ \forall j \in \mathcal{J}, && \text{(18e)}\\
& \sum\nolimits_k a_{j,k} = 1,\ \forall k \in \mathcal{K}, && \text{(18f)}\\
& \sum\nolimits_j \sum\nolimits_{t=s}^{\bar{t}} x_{j,k,t} \leq n_k, \forall k \in \mathcal{K}, && \text{(18g)}\\
& a_{1,1} = a_{J,1} = 1, && \text{(18h)}\\
& \text{(9a), (9b), (9c) and (9d)}
\end{aligned}$$

Clearly, the optimization problem (18) is linear integer problem, which is NP-complete with the computation complexity of $\mathcal{O}(2^{JKT_{J,\max}})$. Fortunately, we can reduce the complexity with the help of *one-time-execution policy* from $\mathcal{O}(2^{JKT_{J,\max}})$ to $\mathcal{O}(JKT_{J,\max})$. It can be solved with optimization solvers, i.e., Gurobi*.

V. Simulation

In this section, we evaluate the system performance numerically under various system setups. We consider topology of the MEC network as a circle area with radius $r = 10$m, where the servers K=9 are homogeneously distributed while the device is located in the centre of the area. We assume the channel bandwidth for the wireless transmission B=50 Mhz with 2.1Ghz carrier frequency. Considering the practical environment, we adopt a path-loss model in [15], given by $PL = 17.0 + 40.0 \log_{10}(r)$. Furthermore, we set the transmit power of the device $P_1 = 10$dBm, the transmit power of servers to $P_k = 32$dBm, $k \in \{2, 3, \cdots K+1\}$ and the noise power to $\sigma^2 = -174$dBm/Hz. For the wired link among servers, we set the transmission rate to $R_{lk} = 100$Mbps, $l, k \in \{2, 3, \cdots K+1\}$ and $l \neq k$. The CPU-cycle frequency of servers are set to $f_k = 3.2$GHz uniformly and the available VM $n_k = 1$. In addition, following [**?**] we set $\kappa = 10^{-11}$ for the energy consumption . Moreover, we set total number of the tasks in the application $J = 10$, the data size of the results for each task $d_j = 10$Mb, $\forall j \neq J$, and the required workloads $c_j = 100$Mb, $\forall j \neq \{1, J\}$. As the device only uploads the local data and downloads the final results, the first and the last task requires no computation, i.e., $c_0 = c_J = 0$. Finally, in our simulation, we consider three different DAG structures as showed in Fig. 1, which are a series structure, a parallel structure and a branch structure. In particular, the

*Gurobi is a commercial optimization solver for linear programming. Available: http://www.gurobi.com.

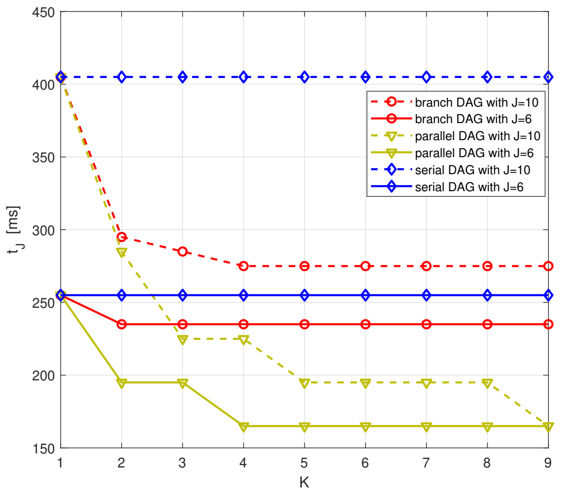

Fig. 2. The completion delay t_J versus the number of servers K under variant DAG setups.

branch structure we considered is a symmetric binary tree graph [11], which is a spacial case of the general structure in Fig. 1-(c), where each task is interdependent with only two successor tasks.

We start with Fig. 2 to compare the performance of three structures of DAG while varying the number of the servers K. To show the influences of the number of the tasks in the application, we evaluate the case with $J = 10$ and the case with $J = 6$, which is equal to the half of the computation tasks in the branch DAG with $J = 10$. Clearly with only one server, the completion delay t_j is the same under all structures. In addition, the completion delays of the serial DAG tasks are constant in M as for serial DAGs tasks can not be computed in parallel. On the other hand, the completion delay of a set of parallel DAG task is significantly reduced by increasing K, when K is small. On the other hand, when the K is relatively large, the delay cannot be further reduced by purely increasing the server number K. In this case, the system already provide sufficient number of servers, while the bottleneck is the computation capability of the server which computes multiple tasks that can not been offloaded to other nodes due to dependency and transmission time issue. Moreover, the delay performance of the branch DAG tasks is in between of the serial DAG and the parallel DAG tasks, i.e., it reduces first and then is constant in K. This is because that a branch DAG structure carries both the features of the other two structure, i.e., it has several layers (like the series structure) while the tasks in each layer can be computed in parallel (which is similar to the parallel structure). Therefore, in comparison to the parallel DAG tasks, the branch DAG requires less servers to minimize the total service period (completion delay).

Subsequently, we investigate the effect of the CPU-frequency on the completion delay in Fig. 3. As expected, increasing the CPU frequency improves the system performance regardless of the structures of DAGs. On the other hand, this improvement by increasing the server computing capability becomes small in the higher frequency range, where the bottleneck of reducing the completion delay becomes the transmission delays among nodes. Among these three DAF structures, increasing the server computing capability is more beneficial for serials DAGs and is relatively less beneficial

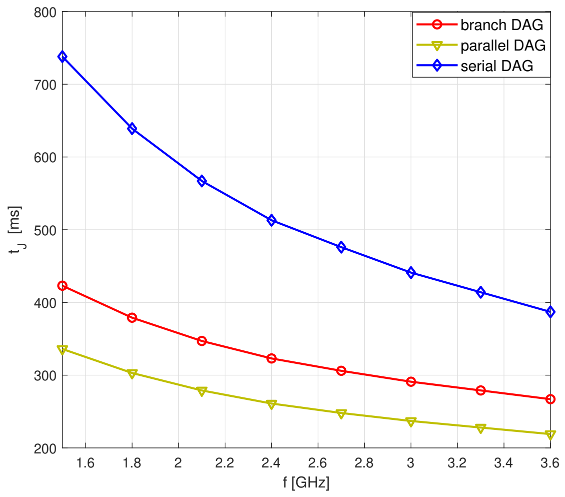

Fig. 3. The completion delay t_J versus the CPU frequency f under variant DAG setups with $J = 10$ and $K = 4$.

for parallel DAGs. Note that the maximal CPU frequency is limited due to the energy consumption and the hardware constraints. Under such case, for parallel and branch DAG tasks, increasing also the number of servers is a more efficient and practical approach rather than purely increasing the CPU frequency of current servers.

Note that each server in the MEC network likely has different energy consumption in a server period, which is different from the centralized server in MCC. Therefore, in Fig. 4, we specifically show the maximum energy consumption over all servers, i.e., $E^{\max} = \max_k\{E_k\}$, when different number of servers are deployed the MEC network. The results are consistent with Fig. 2 that $E^{\max}$ remains the same for the serial DAG due to the *one-climb* policy [17]: the optimal execution with the serial DAG only migrates once between two servers if ever. $E^{\max}$ for both the branch and the parallel DAGs decreases as K increases. Moreover, the parallel tasks costs more energy than the rest two DAG tasks when $K = 1$. Note that for $K = 1$, the process for tasks with different DAG structures are actually the same, thus spending the same amount of energy for the computation. On the other hand,

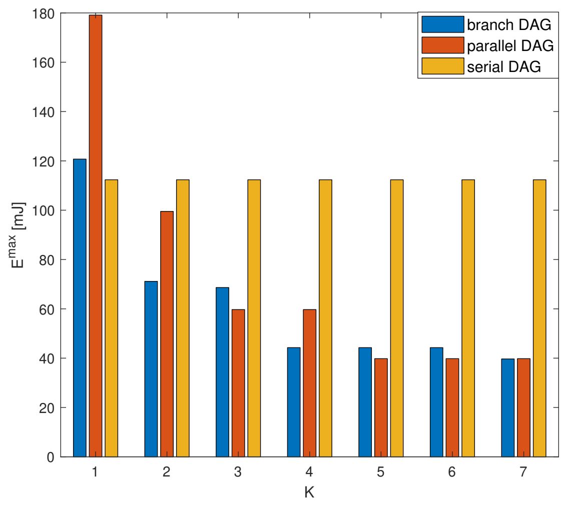

Fig. 4. The maximal energy consumption among the available servers $E^{\max}$versus the number of servers K under variant DAG setups with J=10.

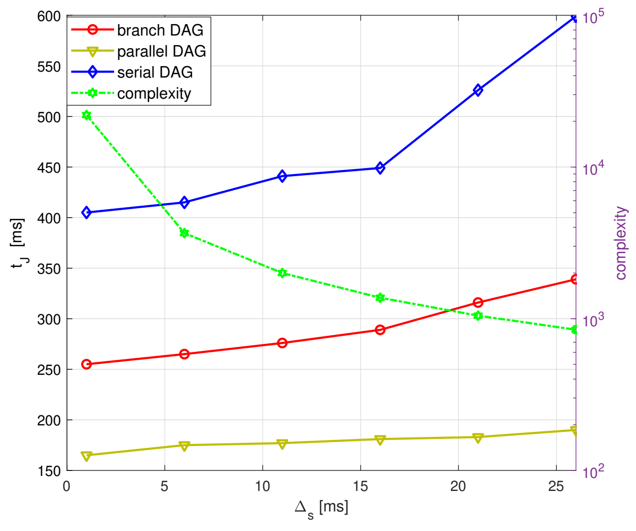

Fig. 5. The completion delay t_J versus the resolution Δ_s comparing the computation complexity of the system under variant DAG setups.

for the parallel DAG task, it requires the server to transmit all the results of each task back to the device, costing a relatively higher communication energy consumption. As a result, the total energy consumption for the parallel DAG tasks are higher than the energy consumption for the serial or branch DAG tasks. As K increases, the energy consumption for communication starts to be split into different servers, $E^{\max}$ is reduced for the both branch or parallel tasks.

Recall that to satisfy the computation capability limits at each server, a set of window check constraints are introduced in (12), where the number of these constraints are decided by the resolution Δ_s. In particular, when $\Delta_s = 1$, the proposed offloading solution is definitely globally optimal, while $\Delta_s = 1$ introduces $\frac{T_{J,\max}}{\Delta_s} = T_{J,\max}$ constraints resulting in a relatively high complexity. On the other hand, if we set Δ_s too large, it causes waste of computation resource and introduces idle delay, as each task is only possible to be computed at the time points $s + w\Delta_s w = 0, 1, ...,$ even though the server is ready a bit before one these time points. In other words, Δ_s actually introduces a tradeoff between the completion delay and the computation complexity. We investigate this tradeoff in Fig. 5, where we set $J = 10$ and $K = 4$. It should be pointed out that we only provide one computation complexity curve, as the complexities for different DAG structure are the same, i.e., the same tasks are computed. It is consistent with our above discussion that the complexity reduces significantly by increasing Δ_s. Meanwhile, the delay performance becomes worse. More interestingly, the serial DAG tasks are most sensitive to the resolution, i.e., relatively high complexity should be spend for such tasks. At the same time, a high Δ_s, corresponding to a relatively low complexity, is acceptable for the parallel DAG tasks, as it provides a competitive delay performance.

VI. Conclusion

In this paper, we studied a MEC network with multiple servers, where servers cooperatively computing interdependent tasks from a user device. After characterizing the energy consumption and the total completion delay, including the communication delay and computation delay, an offloading problem is modeled, which minimizes the completion delay under the energy constraints. To tackle the completion time of each task in each server, we introduced a 3-dimensional decision matrix, so that the optimization problem can be cast as a linear integer programming problem, thus be globally optimally solved. Via the numerical simulation, we investigated the impact of the available servers, CPU frequency on the system performance. In addition, the tradeoff between the delay performance and computation complexity has been addressed.

References

[1] J. Lin, *et al.*"A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications," in *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1125-1142, Oct. 2017.

[2] Z. Chu, F. Zhou, Z. Zhu, R. Q. Hu and P. Xiao, "Wireless Powered device Networks for Internet of Things: Maximum Throughput and Optimal Power Allocation," in *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 310-321, Feb. 2018.

[3] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Netw. Appl.*, vol. 18, no. 1, pp.129–140, Feb. 2013.

[4] Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322-2358, Fourthquarter 2017.

[5] T. Yang, Y. Hu, M. C. Gursoy, A. Schmeink and R. Mathar, "Deep Reinforcement Learning based Resource Allocation in Low Latency Edge Computing Networks," in Proc. *IEEE ISWCS 2018*, Lisbon, 2018, pp. 1-5.

[6] X. Cao, F. Wang, J. Xu, R. Zhang and S. Cui, "Joint computation and communication cooperation for mobile edge computing," in Proc. *WiOpt*, Shanghai, 2018, pp. 1-6.

[7] A. Zamani, S. Shojaee, R. Mathar, A. Schmeink, "Package Assignment and Processing Resource Allocation for Virtual Machines in C-RAN," in Proc. *IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications* (PIMRC 2018), Bologna, Sep, 2018, pp. 1-5.

[8] L. Chen, S. Zhou, and J. Xu. "Computation Peer Offloading for Energy-Constrained Mobile Edge Computing in Small-Cell Networks," in *IEEE/ACM Trans. Netw.* vol. 26, no. 4, pp. 1619-1632, Aug, 2018.

[9] Y. Hu, M. C. Gursoy and A. Schmeink, "Relaying-Enabled Ultra-Reliable Low-Latency Communications in 5G," *IEEE Network*, vol. 32, no. 2, pp. 62-68, March-April 2018.

[10] J. Liu and Q. Zhang, "Offloading Schemes in Mobile Edge Computing for Ultra-Reliable Low Latency Communications," *IEEE Access*, vol. 6, pp. 12825-12837, 2018.

[11] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in Proc. *IEEE INFOCOM WKSHPS*, Toronto, Canada, Apr. 2014, pp. 352–357.

[12] S. E. Mahmoodi, R. N. Uma, and K. P. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Trans. Cloud Comput.*, vol. PP, no. 99, pp. 1–13, 2016.

[13] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in Proc. *IEEE Int. Conf. Comput. Commun.* (INFOCOM), San Francisco, CA, Apr. 2016, pp. 1–9.

[14] T. Q. Dinh, J. Tang, Q. D. La and T. Q. S. Quek, "Offloading in Mobile Edge Computing: Task Allocation and Computational Frequency Scaling," *IEEE Trans. on Commun.*, vol. 65, no. 8, pp. 3571-3584, Aug. 2017.

[15] Y. Corre, J. Stephan and Y. Lostanlen, "Indoor-to-outdoor path-loss models for femtocell predictions," in Proc. *IEEE PIMRC* Toronto, ON, 2011, pp. 824-828.

[16] S. Xi *et al.*, "Real-time multi-core virtual machine scheduling in Xen," in Proc. *2014 International Conference on Embedded Software (EMSOFT)*, Jaypee Greens, 2014, pp. 1-10.

[17] W. Zhang, Y. Wen and D. O. Wu, "Collaborative Task Execution in Mobile Cloud Computing Under a Stochastic Wireless Channel," in *IEEE Trans. on Wireless Commun.*, vol. 14, no. 1, pp. 81-93, Jan. 2015.