Neurocomputing 424 (2021) 160–171

Contents lists available at ScienceDirect

Neurocomputing

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

Brief papers

Joint offloading and scheduling decisions for DAG applications in mobile edge computing

Jie Liang[a,b], Kenli Li[a,b,*], Chubo Liu[a,b], Keqin Li[a,b,c]

[a] College of Information Science and Engineering, Hunan University, Hunan 410082, China
[b] National Supercomputing Center in Changsha, Hunan 410082, China
[c] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Article history:
Received 8 August 2019
Revised 27 October 2019
Accepted 20 November 2019
Available online 15 February 2020

Communicated by Dr. Nianyin Zeng

Keywords:
Mobile edge computing
Task scheduling
Computation offloading
Dynamic voltage and frequency scaling
Makespan

ABSTRACT

Mobile edge computing (MEC) is a promising technology to support computation-intensive tasks for mobile devices which are usually associated with limited resources. Many researches from both scientific and industrial field have put focuses on MEC. However, most of them assume that in a MEC environment, the offloaded tasks are independent or that there is only one server in the MEC center. Nevertheless, in reality, tasks with dependencies take the majority and in a MEC center, there are usually multiple servers. Under this circumstance, previous methods no longer take effects. In this work, we consider offloading with precedence constraints among tasks, and try to minimize makespan over a MEC center with multiple servers. This problem becomes more complex given that a task can not start unless its predecessors are completed. To solve the problem, we jointly involve front end task offloading order and back end scheduling to optimize makespan, and propose a corresponding algorithm called joint re-ordering and frequency scaling (JRFS). Extensive experiments have been conducted. The results show that compared with several other methods, JRFS can achieve better makespan.

1. Introduction

1.1. Motivation

In recent years, mobile devices (MDs) have become an indispensable part of modern life, and have gained increasing popularity. This also brings more demands for computation-intensive and latency-sensitive applications, such as online gaming, gesture and face recognition, and 3D modeling [1–5]. However, MDs are in general resource-constrained, e.g., processing speed, memory size, and battery energy. Mobile Edge computing (MEC) as a new technology provides computing power for mobile devices with limited resources to meet the needs of compute-intensive applications [6]. Compared with cloud computing, MEC has the advantages of reducing latency, saving power consumption for mobile devices, and improving privacy and security for mobile applications.

MEC is a new layer of architecture between mobile devices and cloud computing with the purpose of further enhancing service quality for mobile users [7]. MEC mainly involves two parts, i.e., an access point (AP) and a cloudlet. The latter can be viewed as a small cloud center to serve nearby MDs. Specifically, in MEC environment, MDs can offload their computation-intensive tasks to a nearby AP which covers it. Then, the tasks are executed in the corresponding cloudlet. After execution, the results are returned to the MDs.

Presently, the MEC technique facilitates the service experience of jobs at large-scale low-power devices. Researchers from scientific and commercial fields have been attracted to promote MEC technology. Numerous research work provide overviews of MEC with different aspects including system and network models, optimal control, and multi-user resource allocation [8–11]. About the multi-user resource management and scheduling, existing works only focus on the sequential calls of tasks on the mobile device in the homogenous MEC servers [12–14]. Most of them [14–16] assume that in a MEC environment, the offloaded tasks are independent or that there is only one server in the MEC center. These studies on MEC are still in their infancy [17]. In reality, tasks with dependencies take the majority. How to optimally

* Corresponding author at: College of Information Science and Engineering, Hunan University and National Supercomputing Center in Changsha, Hunan 410082, China.
E-mail addresses: lxj@hnu.edu.cn (J. Liang), lkl@hnu.edu.cn (K. Li), liuchubo@hnu.edu.cn (C. Liu), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.neucom.2019.11.081
0925-2312/

offload an application that are comprised of constraint tasks to heterogenous remote computing servers is a difficult issue, such as DAG-type tasks, because there exist a certain constraint among tasks and constraints among tasks impact the date transmission and job scheduling in the MEC center. For instance, a task must be executed after all its predecessors for DAG-type applications. In order to solve this problem and make a supplement on this issue, we consider offloading with precedence constraints among tasks and try to minimize makespan over a MEC center with multiple servers. On the heterogeneous MEC server side, we use dynamic voltage and frequency scaling (DVFS) technology to select the best execution frequency. None of the above works [14–16] consider both the task precedence and CPU clock frequency of multi-servers MEC system in computation offloading policy. Specially, there are very few solutions proposed to realize DAG tasks scheduling under multiple MEC servers heterogeneous environment, especially combined with DVFS technology for makespan consideration. In this paper, we make some supplementations in this area to some extent.

1.2. Related work

MEC provides cloud-like capabilities to address computation-intensive application, which reduce latency and prolong the battery lifetime of MDs [22], which attracts a large number of research from industry [23,24] and academia [22,25,26].

During the past several years, many previous works focus on optimizing the MEC performance, such as offloading schemes in MEC, resource management and scheduling policies for simple single-user MEC system [27,28] and multi-user MEC system [13,14]. In the single MEC system, Mahnmoodi et al. [27] and Hong et al. [28] comprehensively optimized computational latency and power consumption, taking into account the reduction in average execution overhead. In [13], Mahmoodi et al. further adopted JSCO algorithm for multi-component dependency applications. In [14], a heuristic algorithm was proposed to deal with multi-user computation partitioning problem for multi-user MEC system. Different levels of priorities among multi-users are also considered in [12], where the sequential calls of functions in the MD are considered and the execution order is adaptively adjusted during the server computing process.

For the multi-user system, there are dependencies among tasks which seriously affect the process of executing and offloading in many applications. For example, in an application, the output of some components is the input of other components, then the order of execution of these components cannot be arbitrarily selected. As another example, due to software or hardware limitations, some computing components need to be offloaded to the remote server for calculation, and some can be performed locally. In this case, the task model that considered task precedence will be more complicated than the one mentioned earlier, which should capture the dependencies among different computing tasks in the application [29]. Directed acyclic graph (DAG) is one of such task models. In [18,20,21], the application model is task-call graphs, which is used to specify the constraints and dependencies. Given an graph-type application, Jia et al. presented a heuristic program portioning scheme to offload tasks to cloud for calculation [20]. Kao et al. find optimal task assignments for local and remote devices for dependency graph applications [18]. A dynamic offloading and scheduling strategy is presented to reduce the overhead of power consumption, under the constraints of deadlines for execution time and priority of the task [21]. These works discussed above are for the single MEC server. There do also exist works referring to multiple MEC servers environment [30,31]. Particle swarm optimization is an efficient heuristic algorithm based on evolution intelligence [32], it has been adopted to solve all kinds of optimization problems [33]. For the moment, there have been more literatures on MEC research, Guo et al. design a suboptimal algorithm named as hierarchical GA and PSO-based computation algorithm to optimize computation offloading model [34]. Rodrigues et al. propose a heuristic algorithm called Particle Swarm Optimization, to balance the workload in MEC [35].

Some of multiple MEC servers system are in the homogeneous environment, and the execution speed of MD and MEC server is fixed. However, in a heterogeneous environment, the execution frequency and speed can be different according to the calculation requirements. As the CPU frequency increases, the CPU power consumption will increase linearly. In the heterogeneous environment, some techniques can be adopted to obtain a desired performance by adaptively adjusting its voltages and clock frequencies. Dynamic voltage frequency scaling (DVFS) is one of techniques that are usually used to optimize energy consumption or computing cost. Therefore, combining DVFS into computation offloading and remote execution can provide more flexibility for multiple MEC server system. For example, transmission power and calculation speed [15], application execution time [21] and energy cost [16] are optimized by considering the clock frequency of mobile devices on the MD side. In the literature [15], the calculation speed and the transmission power of the MD are optimized by the DVFS technology. By considering the local computing clock frequency and the remote transmission power allocation, both power cost and execution time are reduced [21]. In [16], the problem of conserving energy of applications in mobile devices is studied by dynamically setting CPU clock frequency on the MD side. However, these works mentioned in above tend to reduce the power consumption by adjusting the CPU clock frequency via DVFS technique on MD side, and performance optimization via DVFS technology on the MEC side is ignored. In the multiple heterogeneous server MEC system, different MEC servers have different set of frequencies, DVFS can be used to adjust CPU clock frequency to make the application execution time and power consumption smaller, and the edge computation resource can be efficiently utilized.

Table 1 summarizes the comparisons between our work and the related works. As shown in Table 1, none of current works simultaneously involve task precedence for MD applications and execution frequency scaling for multiple MEC servers. Specially, no existing work addresses the makespan minimization problem for the DAG applications via considering task constraints in offloading and consider frequency scaling in remote execution through DVFS technology on the multiple heterogeneous MEC side. When incorporating DVFS technology on MEC server side, the problem becomes even harder. To the best of our knowledge, for such category of issue, this is the first dynamic offloading and resource scheduling work that minimize application completion time (makespan) for DAG-type task by considering the CPU clock frequency control in mobile edge computing.

In this paper, we assume that there is one DAG type task in each MD, and each task is totally offloaded to the MEC server for execution base on the premise that the precedences among tasks can be satisfied. If the task is partially offloaded, the constraint relationship among DAG tasks will increase the computational complexity and increase the calculation time. Therefore, we assume that computational resource is sufficient at the MEC server and the tasks in the MD are all offloaded to the MEC for execution for simplicity and better results. MEC servers are heterogeneous, each server has a set of supply voltages and a set of corresponding frequencies. The voltages and corresponding frequencies are discrete. Each server supports DVFS regulation, which means that each server can run at different speeds with different frequencies. Under this consideration, the execution cost of DAG task can be minimized by adjusting the execution frequency and offloading tasks sequence order.

Table 1
Comparisons between JRFS and the related existing schemes.

Schemes	Architecture characteristic(s)	Dependency among tasks	DVFS technique(s)	Objective(s)	Main technique(s)
Kao et al. [18]	Multiple MDs and single MEC server	Subset of DAG	None	Latency minimum subject to a cost constraint	Fully polynomial time approximation scheme
Ra et al. [19]	Two different MDs and single MEC server	General	None	Makespan & throughput	Greedy heuristic
Jia et al. [20]	Multiple application users and single MEC server	Linear and general	None	Completion time of the application & Parallelism between MD and MEC	Load-balancing heuristic
Yang et al. [14]	Single MD and single MEC server	None	Control CPU clock frequency in local computing (MD)	Average application delay	Heuristic
Wang et al. [15]	① Single MD and single MEC server ② One MD and Multiple MEC servers	None	Control CPU clock frequency in local computing (MD)	Energy consumption & latency	Variable substitution technique & univariate search technique
Zhang et al. [16]	Single MD and single MEC server	None	Control CPU clock frequency in local computing (MD)	Energy consumption	Lagrangian multiplier method
Guo et al. [21]	Multiple MDs and single MEC server	DAG	Control CPU clock frequency in local computing (MD)	Energy consumption &application completion time	Heuristic
JRFS	① One MD with DAG-type task and multiple heteregeous MEC servers ② Full offloading	DAG	Control CPU clock frequency in mobile edge computing (MEC)	Makespan	Heuristic

1.3. Our contributions

In this paper, we focus on task sequence offloading under task dependency constraints and computing frequency scaling under multiple MEC servers environment. Specifically, there is one MD which is associated with a DAG-type task. In order to guarantee priority, all of tasks are offloaded to remote execute according to its own priority. Clock frequency were adjusted and compared step by step on the MEC side, the optimal frequency configuration were drew for *Makepan* minimum application execution. The aim of our work is to find an optimal scheduling mechanism which guide the tasks offloading under tasks dependency constraint and the setting of the execution frequency of tasks on the MEC server, such that *Makespan* of application is minimized. Main contributions are listed as follows.

- We propose a heuristic algorithm for offloading DAG applications and scaling frequencies on the multiple MEC servers environment. Under this circumstance, we jointly involve front end task offloading order and back end scheduling to optimize makespan.
- We involve uneven arrivals and tasks precedences on MD to decide mapping and execution order on the front end. The effect of offloading transmission is added to the traditional HEFT algorithm, which is also an application supplement of HEFT algorithm in the mobile edge computing.
- We involve DVFS technique for DAG application executed on the back MEC server end. A desired performance is obtained by adaptively adjusting its different clock frequencies in the heterogeneous environment. To our knowledge, other literatures tend to optimize performance via DVFS technique on MD side. Few works can be found for this on MEC server side. This work may also make some supplementations in this area to some extent.

We perform extensive random experiments. The results show the capability of the proposed algorithm in achieving a smaller makespan compared to several other heuristics.

1.4. Paper outline

The rest of this paper is organized as follows. In Section 2, the models and problem formulation are presented. In Section 3, we propose two heuristic algorithms. Extensive random experiments are tested and the results of the experiments are analyzed in Section 4. The work is summarized in Section 5.

2. Models and problem formulation

This section introduces the system model used in this article, including the computational task model and the offloading execution model.

2.1. Architecture model

The architecture of a single-user MEC system is presented in Fig. 1. It contains two basic components: (1) a mobile device connected to the network where user has a DAG-type task to be executed. (2) MEC server which is a center that can provide remote computing services. The MEC server is placed at the wireless access point (AP), and can communicate with mobile devices through the wireless channel, and perform remote operations to calculate computing tasks on mobile devices. M is the number of MEC servers and $\mathcal{M} = \{1, \ldots, M\}$ be the corresponding servers set. Each server has a set of supply voltages and a set of corresponding frequencies, which are denoted as $\mathcal{V}_j = \{v_{j,1}, \ldots, v_{j,m_j}\}$ and $\mathcal{F}_j = \{f_{j,1}, \ldots, f_{j,m_j}\}$, respectively, where m_j is the number of voltage (frequency) levels of the MEC server j ($j \in \mathcal{M}$).

Tasks on mobile devices need to be transferred to an AP before being offloaded for remote execution, and then the corresponding MEC server perform the task execution. We assume that computational resource is sufficient at the MEC server, the execution delay can be ignored. Because the computing resources in the heterogeneous MEC server system are relatively redundant compared to the computing tasks of the DAG application. It is further assumed that the feedback delay is negligible because output data size is assumed to be small.

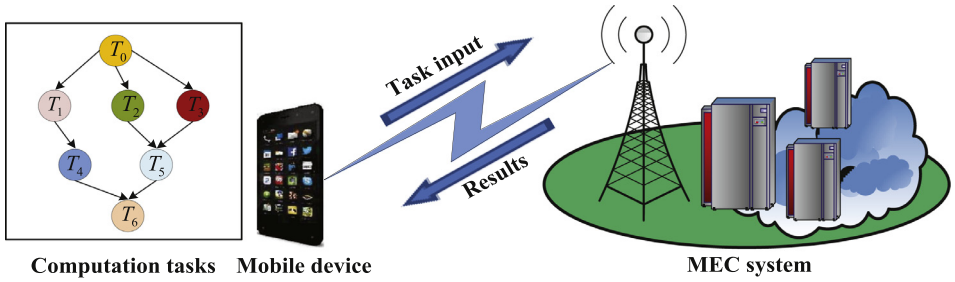


Fig. 1. Architecture model.

2.2. Computation task model

Assuming there are N precedence constrained tasks in the MD that need to be executed. It is denoted as $\mathcal{T} = \{T_1, \ldots, T_N\}$. Each computation task is characterized by a two-tuple of parameters, $\langle d_i, c_i \rangle$, where d_i (in bits) is the amount of the task input data, and c_i (in CPU cycles) is the workload. The values of d_i and c_i depend on the nature of the computation tasks and can be obtained through off-line measurements [36].

Because the outputs of some tasks in an application are the inputs of others, the dependency among different constraint tasks in an application affects the procedure of execution and computation offloading. For capturing the inter-dependency among different computation tasks in an application, a directed acyclic graph (DAG) model is adopted. DAG is a directed graph with no directed cycles. It consists of a set of vertices and directed edges, each of which connects one vertex to the other. DAG model is described by a two tuple $\mathcal{G} = \{\mathcal{T}, \mathcal{E}\}$. Where $\mathcal{T}$ is the task set, $\mathcal{T} = \langle T_1, \ldots, T_N \rangle$, which consists of different associated task in the application. $\mathcal{E}$ is the set of edges between tasks denoting the execution order and communication between two adjacent tasks, which specifies their dependencies. Such as an edge $(i, i') \in \mathcal{E}$ between task nodes i and node i' represents that task T_i is the predecessors of task $T_{i'}$. Namely, task T_i should complete its execution before task $T_{i'}$. The task without any predecessor is called an entry, and the task without any successor is called an exit. The weight attached to each task T_i represents the computation requirement, described by d_i and c_i. The weight assigned to an edge represents the communication cost between two tasks T_i and $T_{i'}$, denoted as $C_{i,i'} = data(T_i, T_{i'})$. The data transfer between two tasks is only required when the two tasks are assigned to different processors in MEC server, namely communication cost is negligible when the two tasks are executed on the same processor. In this paper, all the computation tasks are offloaded to the MEC server for mobile-edge execution.

2.3. Task offloading and remote execution model

Tasks on the mobile device need to be transferred to the MEC server before being executed. Task offloading decision for the N tasks is denoted as $\boldsymbol{\phi} = (\phi_1, \phi_2, \ldots, \phi_N)$. It is a permutation of the task order. ϕ_i is the ith task being offloaded to the MEC system. Hence, $\phi_i \in \{T_1, \ldots, T_N\}$ and $\phi_i \neq \phi_{i'}, i' \neq i, \forall i, i' \in \{1, \ldots, N\}$.

Each MD has an antenna for transmitting data, and only one task is transmitted at each time. The transmission rate of the task is as follows

$$R(p_i) = w\log_2\left(1 + \frac{g_0 (L_0/L)^{\theta} p_i}{N_0 w}\right), \tag{1}$$

where p_i is the task transmission power, g_0 and θ is the path-loss constant and the path-loss exponent, respectively. L_0 is the reference distance, and N_0 is the noise power spectral density (Table 2). For each task T_i, the transmit energy efficiency is convex with the transmit power [37]. Therefore, the optimal transmission rate R_i^* can be obtained by setting the suitable power supply. Then the transmission time can be obtained as

$$\Gamma_i = \frac{d_i}{R_i^*}. \tag{2}$$

Denote φ_i as the sever for executing offloaded task T_i. Then, the execution time of T_i is given as

$$E_i = \frac{d_i c_i}{f_{\varphi_i}}, \tag{3}$$

where f_{φ_i}(in Hz) is the CPU-cycle frequency for executing task T_i.

Table 2
Notation.

Notation	Description
M	The number of MEC server
f_j	Execution frequency of MEC server j
$\mathcal{V}_j, \mathcal{F}_j$	The set of supply voltage (frequency) of server j
m_j	The number of voltage (frequency) levels of server j
c_i	Workload of task i
d_i	Input data of task i
$C_{i,i'}$	The amount of data exchange between task i and i'
ϕ_i	The ith task being offloaded to the MEC system
Γ_i	The transmission time of task i
φ_i	The offloaded MEC server for executing task i
f_{φ_i}	The CPU-cycle frequency of processor executed by task i
$Ppred_{\phi_i}$	The front adjacent predecessor task of ϕ_i on the same MEC server
$pred_{\phi_i}$	Predecessor task of ϕ_i in DAG
$B_{\varphi_i,\varphi_{i'}}$	The average communication bandwidth among servers executing task ϕ_i and $\phi_{i'}$

2.4. Problem formulation

2.4.1. Performance metrics

The schedule length, *makespan*, is chosen to be a metric to evaluate the performance in the mappings for DAG applications in mobile device. It is the complete time of the exit task. For the mobile-edge execution, the start execution time of a task is affected by two factors, the task's ready time and the server's ready time. The former is affected by the DAG constraints and channel transmission, while the latter is decided by the computing power and its current computation amount of server. Specifically, task's ready time is the sum of transmission time of tasks offloaded previously and earliest start time (EST) depending on the earliest finish time (EFT) of the task being offloaded previously (i.e., its predecessor task in DAG). For the task T_{entry}

$$EST_{T_{entry}} = \Gamma_{\phi_1}. \tag{4}$$

For other tasks in graph, the EST of task ϕ_i executed on MEC server can be obtained as

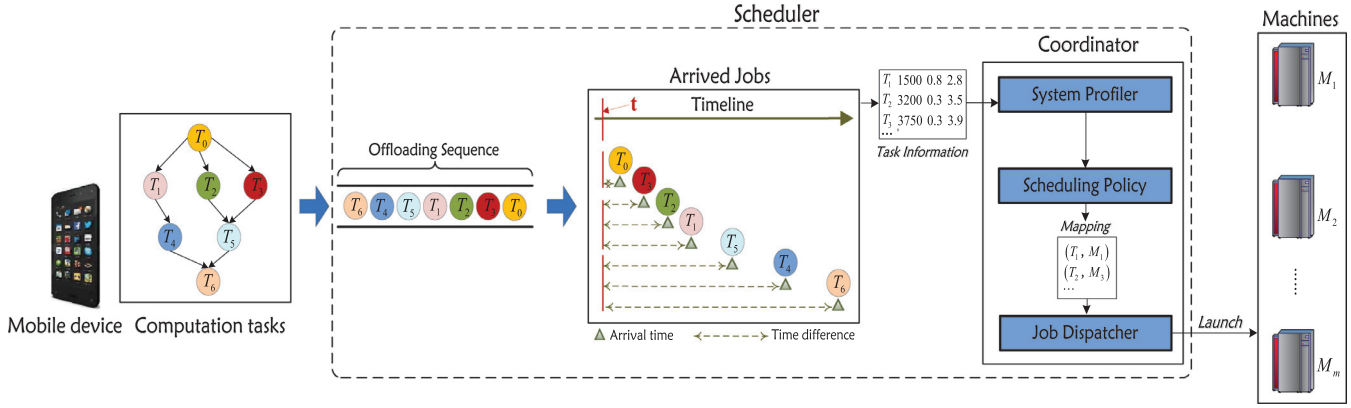


Fig. 2. Scheduling framework.

$$EST(\phi_i, f_{\varphi_{\phi_i},k}) = \max\left\{\sum_{i=1}^{i}\Gamma_{\phi_i}, EST_{Ppred(\phi_i)} + E_{Ppred(\phi_i)}, \max_{\phi_{i'}\in pred(\phi_i)}\left(EST_{\phi_{i'}} + E_{\phi_{i'}}(f_{\varphi_{\phi_{i'}},k}) + \frac{C_{\phi_i,\phi_{i'}}}{B_{\varphi_{\phi_i},\varphi_{\phi_{i'}}}}\right)\right\}, \quad (5)$$

where $Ppred_{\phi_i}$ and $pred_{\phi_i}$ are the front adjacent predecessor task on the same MEC server and a set of direct predecessor tasks of ϕ_i in DAG, respectively. The level of the frequency set by MEC server is k, where $f_{\varphi_{\phi_i},k}$ represents the $k-th$ frequency of server φ_{ϕ_i}. $B_{\varphi_{\phi_i},\varphi_{\phi_{i'}}}$ is the average communication bandwidth among servers executing task ϕ_i and $\phi_{i'}$.

Accordingly, the earliest execution complete time of a task is the summation of its execution time and its earliest start time. Thus, EFT can be expressed as

$$EFT(\phi_i, f_{\varphi_{\phi_i},k}) = EST(\phi_i, f_{\varphi_{\phi_i},k}) + E_{\phi_i}(f_{\varphi_{\phi_i},k}), \quad (6)$$

where $E_{\phi_i}(f_{\varphi_{\phi_i},k})$ is the execution time of ϕ_i scheduled on server φ_{ϕ_i} with the kth level of frequency. The EFT and EST values are computed recursively from the entry task to the exit task. After all tasks in DAG are transmitted to the MEC and mapped on different servers, the time overhead to complete all tasks is defined as *makespan* by

$$makespan = \max\{AFT(T_{exit})\}. \quad (7)$$

2.4.2. Problem analysis

As mentioned earlier, all the MD's tasks are offloaded to MEC server for its execution, and each server can run at different speeds with different available frequency levels (AFLs). During the execution of the task in remote server, execution time of the task is changed by scaling the CPU clock frequency. In the process of computation offloading, the order of task sequence offloaded also affects the mapping relationship on which server the task is scheduled. Thus, these two aspects affect the DAG's makespan. In our computing system, we try to minimize makespan through adjusting offloading order and frequency scaling for mobile-edge execution. Specifically, given a DAG application $\mathcal{G} = \langle \mathcal{T}, \mathcal{E} \rangle$ and a mobile edge computing system, the goal is to offload MD's tasks to different MEC servers with "optimal combination" of execution frequency and offloading sequence, such that its DAG's *makespan* is minimized while guaranteeing the constraint between tasks in DAG application and that a task can only be offloaded to one server, i.e.,

$$\begin{aligned} &\text{minimize} \quad makespan = \max\{AFT(T_{exit})\}, \\ &\text{s.t.} \quad \mathcal{S}_j \cap \mathcal{S}_{j'} = \emptyset, \forall j, j' \in \mathcal{M}, \end{aligned} \quad (8)$$

$$AFT_{T_{\phi_{i'}}}, \phi_{i'} \in Ppred_{\phi_i}, \quad (9)$$

where $\mathcal{S}_j$ is the set of tasks that are offloaded to MEC server j.

3. Scheduling scheme

3.1. Scheduling framework

To find the optimal scheduling solution, the framework implements a centralized coordinator, which includes three modules: system profiler, scheduling policy and job dispatcher. The framework is presented in Fig. 2.

System Profiler receives task information and MEC server current capacity information and give corresponding response to minimize tasks's earliest finish time. It is similar to the processor selection phase in HEFT algorithm [38].

The scheduling process in policy is divided into three phases, task computation offloading, partitioning and frequency scaling. The scheduling scheme at each iteration is obtained through three phases and stored in the scheduling policy pool. In the first phase, there are constraints between the tasks in the DAG-type application. A task must be executed after all its predecessors have executed it. Therefore, these tasks need to be transmitted over the channel to the AP and assigned to the machine on MEC under the DAG constraints. In the second phase, Eq. (5) and (6) are applied to allocate the offloaded task onto the MEC server for *makespan* minimum. After the initial mapping of the entire DAG application, in which MEC servers each task is assigned to execute is determined. In the third phase, we scale the CPU clock frequency of the server executed by the offloaded task to further reduce *makespan*.

Through transmission of the task sequence, the selection of processors, the scaling of the frequency and re-ordering the sequence, a scheduling strategy scheme is finally obtained, labeled as $\mathcal{M}(\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathfrak{F})$. $\mathcal{S}$ and $\mathcal{F}$ are respectively the final sequence offloading order and initial frequency of servers. $\mathcal{P}$ is selected mapping relationship between processors and tasks. $\mathfrak{F}$ is the final execution frequency of servers executed by offloaded tasks, which is obtained in frequency-scaling algorithm. We obtain one scheduling mechanism at each iteration and this scheduling strategy is stored in the scheduling policy pool. By continuous iterative operation, another scheduling policy is obtained and stored in the pool. Finally, the scheduler selects the optimal scheduling strategy from the scheduling policy pool and dispatches tasks to the corresponding processors for execution with selected frequency under the particular offloading order.

3.2. Optimization of task offloading scheduling

In this work, task relationship in MD is described by DAG. At the beginning, we perform the initial arrangement of the task offloading sequence according to the constraint relationship among

DAG tasks. Tasks are initially ordered by their offloading priorities that are based on *upward* ranking starting from the exit task to reversely traverse the DAG graph. Based on the generated offloading sequence with this method, the task that is assigned with the highest priority is ready to start earliest.

The *upward* rank of a task is recursively calculated by

$$rank_u(T_i) = \overline{c_i} + \max_{T_{i'} \in succ(T_i)} (\overline{C_{i,i'}} + rank_u(T_{i'})), \quad (10)$$

where $\overline{c_i}$ is the average execution cost of a task T_i, $succ(T_i)$ is the set of immediate successors of task T_i, and $\overline{C_{i,i'}}$ is the average communication cost of edge (i, i'). The upward rank value is the length of the critical path starting from that task to the exit task, which is the maximum of the summations: successor execution time, communication time of each edge, and the weight of each node. After task-prioritizing, we offload the task with the largest rank value first. By decreasing order of upward rank value, the task offloading sequence meeting the constraints of the DAG is generated. It presents a topological order of tasks [38], which meet the constraints of the DAG task.

When the task is achieved with topological offloading order, the task is assigned to the "best" processor according to EFT minimum principle combining with task and machine information [39], as shown in Eqs. (5) and (6). The earliest available time of a processor for a task execution is the task's arrival time or the time when processor completes the execution of its last assigned task. In this paper, it is assumed that one mobile device can only occupy one subcarrier of an AP and only one task can be transferred at a time. Besides, the scheduler assigns the computation task with a *first-come-first-serve* (FCFS) fashion. The order in which tasks are executed is the same as the order in which tasks are offloaded. When the order of tasks offloaded is different, the order of the task reaching changes, and the mapping relationship between task and server is different. Accordingly, the makepan will be affected. Therefore, we can adjust the order of task offloading to bring optimization for makespan.

Different offloading sequences lead to different task execution orders, which affects makespan. In order to reduce the completion time of application, to adjust the order of offloading is one of the optimization methods. When the start time of an offloaded task depends on its arrival time, this means that the task can be selected in the process of sequence re-ordering. The mapping relationship between task and server can be changed by adjusting offloading order of this task, resulting in influence of makespan. In this case, the selected task is moved one step forward in the original offloading sequence. And then we make a judgement on whether the new offloading sequence satisfies all the partial orders of the DAG. If it is satisfied, the offloading sequence is re-ordered. Otherwise, the re-ordering cancels. The re-ordering mechanism is based on the judgment of the execution start time of the task, resulting in a new offloading sequence. This is a guided and directed sequence re-ordering technique, which guides a certain task to adjust the location and derives a new offloading sequence.

As mentioned above, the offloading sequence that satisfies the DAG constraint is not unique. Therefore, a large number of offloading sequences under the DAG constraint can be generated randomly, and then these sequences are initialized and offloaded. This is another way to affect makespan by re-ordering the offloading sequence. The output is obtained through partitioning and frequency scaling. Finally the offloading sequence that produces the minimal makespan is found in these sequences. This is a random re-ordering approach.

3.3. Optimization of execution frequency selection

Based on the generated execution schedule from the phases of task computation offloading and partitioning, each selected task is scheduled to the server which leads to earliest finish time. We all know that, *makespan* is dominated by the start time and execution time of exit task in the DAG application. The start time of exit task is dominated by one of $T_{exit}{}'$ predecessors or its arrival time. Similarly, the start time of certain task is dominated by its predecessor or its arrival time and so on until the entry task. Hence, we can firstly scale the execution frequency of the server executed by exit task to decrease *makespan*.

To further minimize the execution cost, we adjust the execution frequency of the server that executed the task, starting from the processor where the exit task is located to reversely traverse the DAG graph. Before each step of frequency scaling, the constraints between tasks need to make a judgment that if the start time of task is dominated by the arrival time or its predecessor. If the start time of task is affected by its predecessor, we adopt the frequency scaling approach to further decrease *makespan* and record the current smaller makespan.

As shown in Algorithm 1 , at each iteration of frequency-scaling, we firstly adjust the frequency from the exit task at the bottom of the DAG, and scale its execution frequency one level up and the corresponding decreased makespan resulted by this scaling operation is recorded. As first step, we find the exit task's predecessor that dominates the exit task, and scaling its execution frequency level when it has an influence on makepan. If the start time of task is dominated by itself, namely its arrival time, the frequency scaling process can not perform. To use an analogy, we do similar operations till there does not exist such a task that can further scale frequency of its server or find a certain task that its execution time depends on itself. The frequency scaling process terminates.

From the above, it can be seen that the goal is to optimally set the clock frequency of the MEC server selected to minimize the completion time.

3.4. The makespan minimization algorithm

In the first proposed algorithm, the offloading sequence adjustment and the task's execution frequency scaling are alternately updated. The key steps of this algorithm are demonstrated in Algorithm 2 . As shown in Eq. (5) in Section 2, it can be seen

Algorithm 1: Frequency scaling algorithm.

```
Input: Available frequency levels for each processor m_τ={m_0, m_1, ..., m_j}, (j = M − 1);
φ_i ← φ_TaskNum;
while true do
    // The level of the frequency for task i set by MEC server is k;
    if k < m_j then
        f_{φ_{φ_i},k} ← f_{φ_{φ_i},k+1};
        update AFT of the exist task;
    end
    else
        φ_i ← i' ∈ pred(i) which dominates the start time of task i;
        continue;
    end
    if EST(φ_i)==ArrivalTimeφ_i then
        return index ← i;
        break;
    end
end
Return -1;
```

Algorithm 2: Joint re-ordering and frequency scaling algorithm (JRFS).

Input: the number of re-ordering $Threshold_Reorder$, G(V,E), Available frequency levels for each processor m_τ={m_0, m_1, ..., m_j};
//Initialize the transmission order based on the priority of the rank value;
while $Threshold_Reorder > 0$ **do**
  for variable i from 1 to $TaskNum$ **do**
    $\phi \leftarrow Order[i]$;
    for each processor P_j in processor set P **do**
      Calculate $EFT(\phi, f_{\varphi_\phi}, k)_{p_j}$;
    end
    Schedule Task ϕ on processor P_j such that $EFT(\phi, f_{\varphi_\phi}, k)_{p_j} \leqslant EFT(\phi, f_{\varphi_\phi}, k)_{p_q}$, $\forall P_q \in P$;
  end
  FrequencyScaling();
  $i \leftarrow FrequencySacling()$;
  Record the updated $Makespan$;
  Record the current scheduling scheme $\mathcal{M}(\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathfrak{F})$;
  if $i! = -1$ **then**
    Adjust the transmission order of task i under the DAG constraint;
  end
end
Select the scheduling scheme and transmission sequence that the makespan is minimum;

Algorithm 3: Hierarchical re-ordering and frequency scaling algorithm (HRFS).

Input: Number of transmission sequences N, Number of tasks $TaskNum$;
while $N > 0$ **do**
  //Generate a DAG topology sort sequence as the transfer order;
  $Order(\tau) = \{\phi_1, \ldots, \phi_{TaskNum}\}$;
  //Initial scheduling based on the EFT minimum principle according to the transmission sequence;
  for variable i from 1 to $TaskNum$ **do**
    $\phi \leftarrow Order[i]$;
    for each processor P_j in processor set P **do**
      Calculate $EFT(\phi, f_{\varphi_\phi}, k)_{p_j}$ according to Eq.(6);
    end
    Schedule Task ϕ on processor P_j such that $EFT(\phi, f_{\varphi_\phi}, k)_{p_j} \leqslant EFT(\phi, f_{\varphi_\phi}, k)_{p_i}$, $\forall P_i \in P$;
  end
  //Frequency Scaling Algorithm;
  FrequencyScaling ();
  Record the updated $Makespan$;
  Record the current scheduling scheme $\mathcal{M}(\mathcal{S}, \mathcal{F}, \mathcal{P}, \mathfrak{F})$;
  $N--$;
end
Select the scheduling scheme and transmission sequence that the makespan is minimum;

that the execution time of the task depends on frequency level on the scheduled processor. Accordingly, the mapping relationship between task and processor will also be different with different initial frequencies. In this paper, middle frequency levels are set as initial frequency to execute processor selection phase.

At first, initial offloading sequence is generated based on upward ranking value starting from the exit task to reversely traverse the DAG graph. Tasks in DAG are initially transmitted and offloaded by their priorities. And then the tasks offloaded are scheduled to MEC server by the minimum EFT principle, resulting in that which server each task is assigned to is determined. In the frequency-scaling phase, we adjust the execution time or start time of task through frequency scaling to decrease makespan until the entry task or certain task that its EST is dominated by arrival time, starting from the exit task to reversely traverse the DAG graph. When the start execution time of certain task is dominated by its predecessor, we adjust the frequency of the server executed by the task. While when the start time of an scheduled task depends on itself, we invoke the re-ordering offloading sequence, the offloading sequence is rearranged to adjust the arrival time of the task for makespan decrease. The new offloading sequence is generated, which is guided by the frequency scaling process. In this algorithm, these two processes interact with each other.

Through the transmission of the task sequence, the selection of processors and the scaling of the frequency, one scheduling strategy scheme $\mathcal{M}$ is finally obtained. At each iteration, the scheduling strategy obtained is stored in the scheduling policy pool. Finally, the scheduler selects the optimal scheduling strategy from the scheduling policy pool and dispatch tasks to the corresponding processor with selected frequency under the particular offloading order. This is a algorithm that integrates adjustment of offloading sequence and frequency scaling in the scheduler. The idea is visualized in Algorithm 2.

Taking the DAG in Fig. 1 as an example, Fig. 3 presents corresponding illustration for the process of adjustment of offloading sequence and frequency scaling in the scheduler. As shown in Fig. 3, the tasks are initially offloaded in descending order of rank values. Then with the principle of minimum execution finish time, the processor are selected for each task according to the task's information. We assume that the execution start time of T_6 is governed by the its arrival time. After the initial mapping of the DAG application, we scale the execution time of T_7 to decrease *makespan*. Notice that, the start time of T_7 is dominated by one of its predecessors T_6. Hence, we scale the execution frequency of T_6 one level up to impact the *makespan*. However, the start time of T_6 is dominated by its arrival time, not by its predecessors. Therefore, this frequency-scaled path is interrupted. We turn to re-order the offloading sequence by adjusting the position of T_6 in the transmission sequence, T_6 is moved forward one step to the front of T_5. Then the new offloading sequence that meets all the DAG constraints continues to be offloaded and scheduled.

In addition, we proposed another algorithm where adjustment of offloading sequence and execution frequency are two separate independent processes, as shown in Algorithm 3. The order of offloading sequence is randomly generated at each time, and the results of frequency scaling have no effect on the adjustment of offloading sequence order. At each iteration, the frequency scaling is terminated till it can not continue, at this time the scheduling scheme and the corresponding makespan are recorded. And then another offloading transmission sequence is randomly generated, the remaining phases are executed in the same manner above, and another scheduling policy is obtained and stored in the pool. Finally, the optimal scheduling strategy among all scheduling strategies is selected and tasks are dispatched to the corresponding processor with selected frequency under the particular offloading order.

Whichever of the two algorithms, the ultimate goal is make makespan minimum under the DAG constraint based on task offloading scheduling and execution frequency selection. Specifically, the EFT minimum principle is used to impact the mapping relationship between tasks and servers by adjusting the sequence

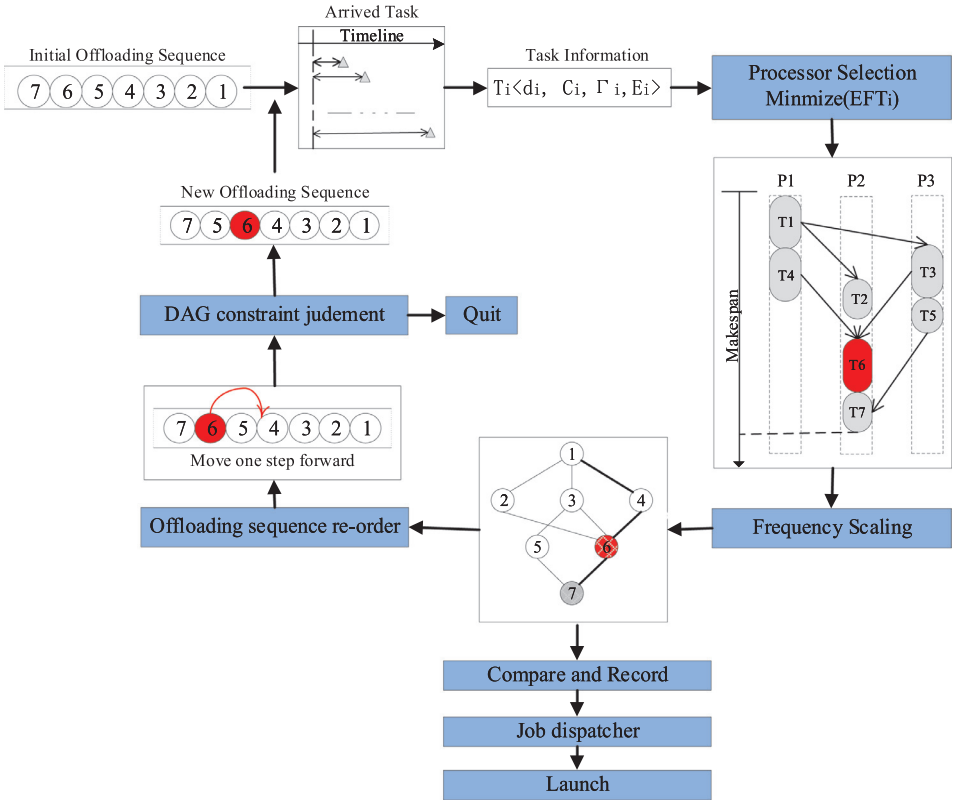


Fig. 3. Scheduler together with frequency scaling and offloading sequence re-order.

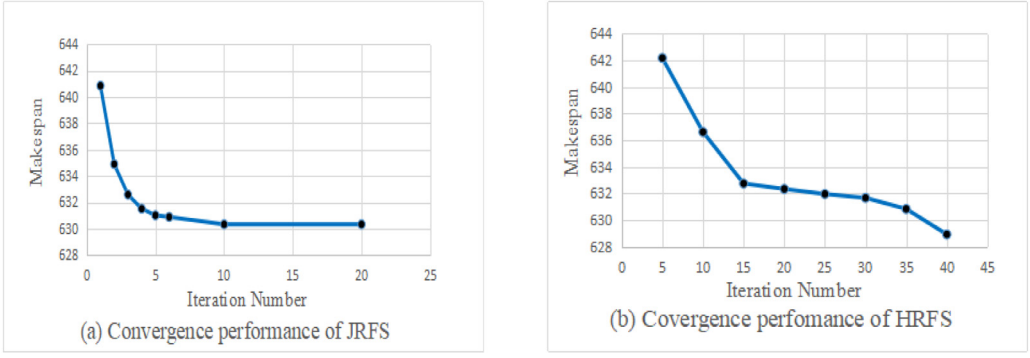


Fig. 4. Convergence performance of the two proposed algorithm.

offloading order, meanwhile the DVFS technology is adopted to scale the frequency of the server executed by tasks.

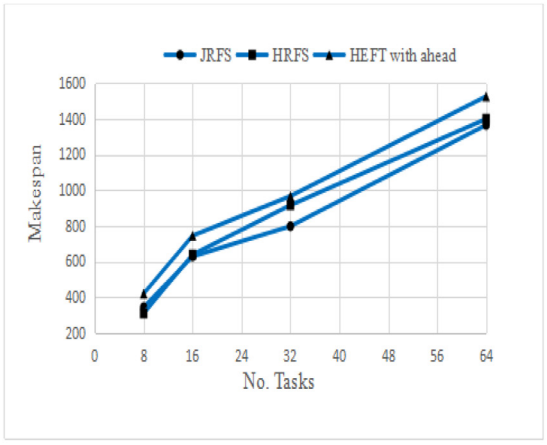
4. Experiment evaluation

The performance of the proposed two algorithms are evaluated and discussed. To our knowledge, few works that consider both the task precedence in MD and CPU clock frequency of MEC server in computation offloading policy can be found. Hence, we implement two heuristics and compare the two algorithms. In addition, we also compare the performance of the JRFS scheme with the HEFT with a lookahead in [40]. To further understand the adaptability and scalability of our proposed algorithm, we considered some randomly generated DAGs whose degree of parallelism and computation/communication ratio could be controlled. Finally, we made real data measurements based on two real applications, FFT and gaussian elimination.

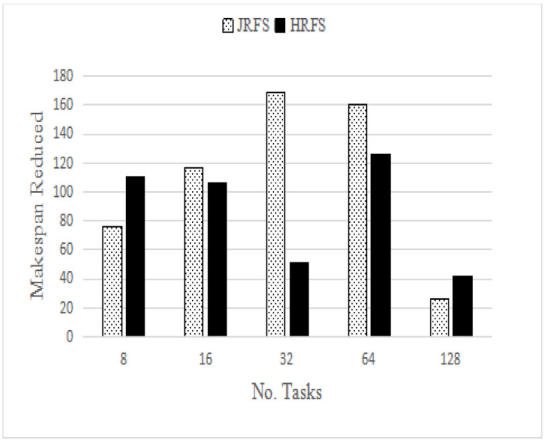
In the proposed two algorithms, the number of iterations is the number of sequence re-ordring for JRFS and the number of sequences generation for HRFS. To show the overall convergence of two algorithms, we display the convergence evolution of two algorithms. In Fig. 4, it can be seen that as the number of iterations increases, the makepan gradually decreases. For JRFS, when the number of iterations reaches a certain amount, makepan does not change. This time indicates that the optimization effect of the JRFS algorithm reaches the maximum. When the number of iterations for HRFS increases gradually, it will tend to the effect of JRFS, and as the number of iterations increases, the makepan will gradually decrease. JRFS gained a better makepan while sacrificing computational complexity.

4.1. Simulations for the random DAG application

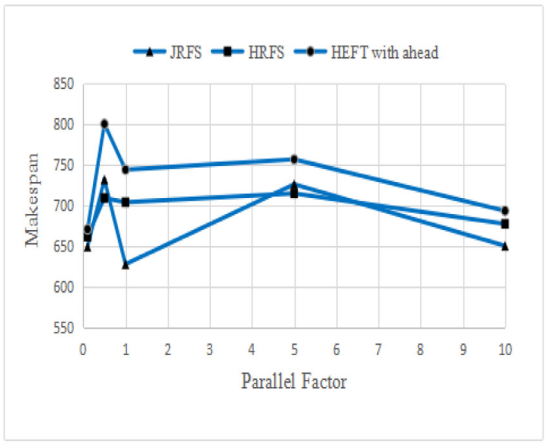
In the following, to evaluate the detail of the two algorithms, we implemented a directed acyclic taskgraph generator. In the ran-



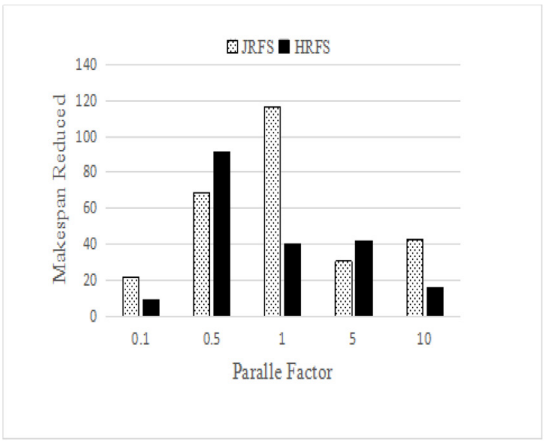
(a) 16 P, $\lambda = 1$, CCR=1



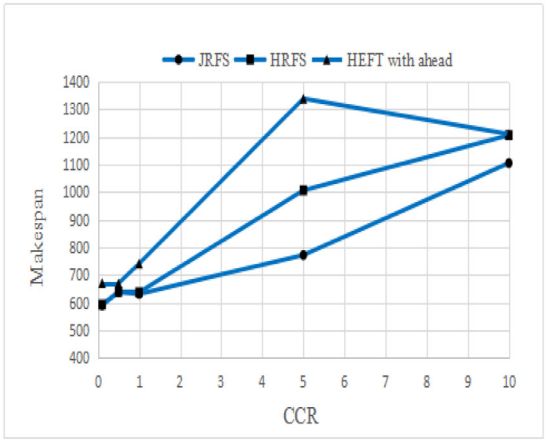
(b) 16 P, $\lambda = 1$, CCR=1



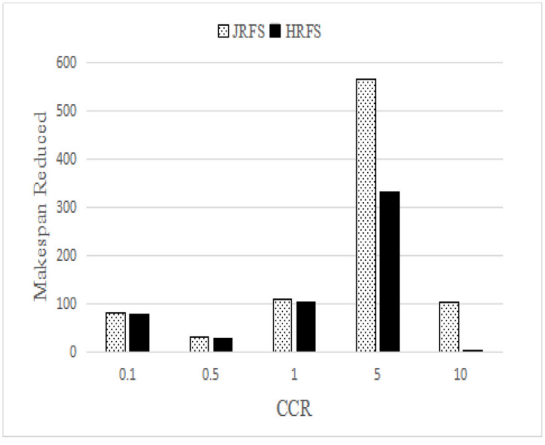
(c) 16 P, 16 Tasks, CCR=1



(d) 16 P, 16 Tasks, CCR=1



(e) 16 P, 16 Tasks, $\lambda = 1$



(f) 16 P, 16 Tasks, $\lambda = 1$

Fig. 5. Random simulation results.

dom DAG experiments, graph are generated based on the various fundamental input parameters. The first parameter is the communication to computation time ratio (CCR). It is computed by the average communication cost divided by the average computation cost of an application DAG. The second characteristic is its parallelism factor, which is the height and the width of an application DAG. And the third is DAG size, which is the number of DAG nodes. we implemented a set of experiments to evaluate the effect of the taskgraphs characteristics on the completion time of the application for different algorithms. We compare the performance of JRFS and HRFS algorithms against the HEFT with lookahead [40]. In each simulation experiment set, there is only one variable, and the other parameters are fixed constants.

Fig. 5 (a) shows makespan against the number of tasks in the DAG. The performance of the two proposed algorithms is better than the HEFT with a lookahead algo-

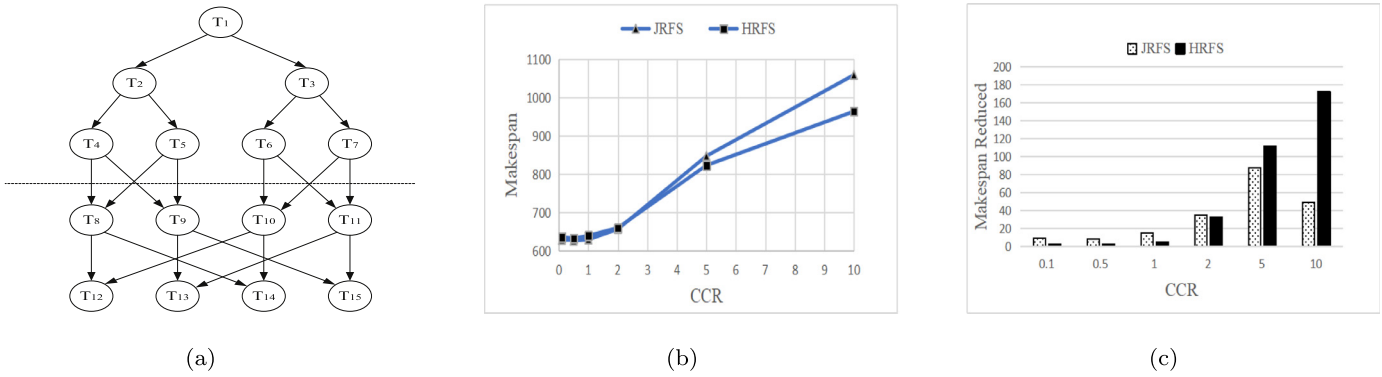


Fig. 6. Results for the FFT graph.

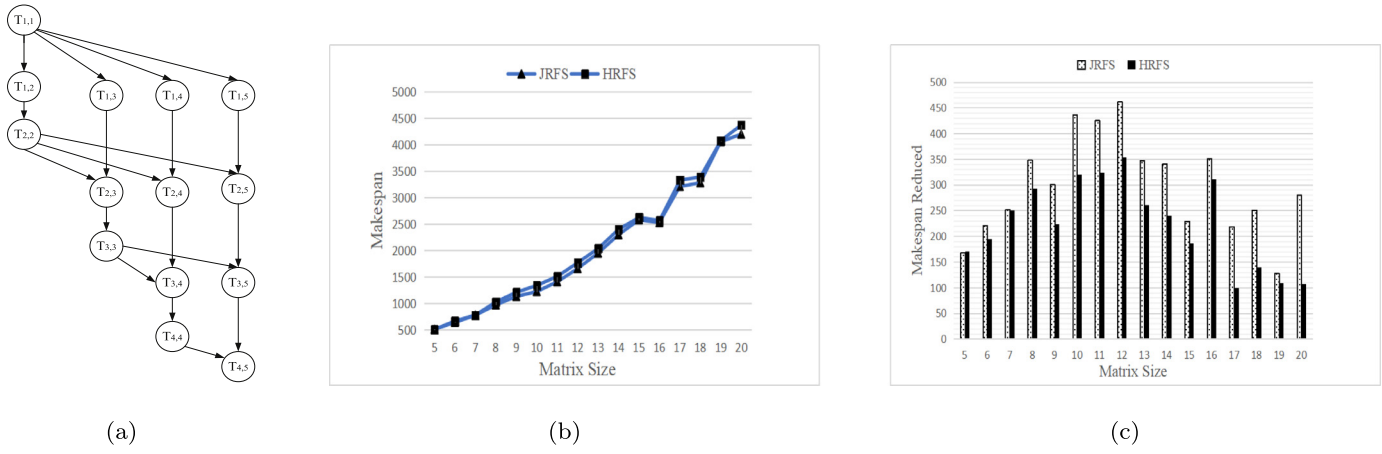


Fig. 7. Results for the Gaussian elimination.

rithm. The results of the makespan reduced that comparing the two algorithms to HEFT with a lookahead are shown in Fig. 5(b). Compared with the HEFT with ahead algorithm, the JRFS algorithm has the best optimization when the number of tasks is equal to 32. The HRFS algorithm works best when the number of tasks is equal to 64. Optimization of both algorithms reduce in a large taskgraph, such as 128 tasks. That is because as the size of the task graph increases, the tasks to be unloaded are more dispersed, and the frequency scaling and sequence re-ordering are affected, resulting in performance degradation.

Figs. 5 (c) and (f) show makespan against the parallel factor and CCR of the DAG, respectively. The performance of the two proposed algorithms are better than the HEET with a lookahead algorithm. In Fig. 5(d), it can be found the JRFS algorithm has the best optimization when the parallel factor is equal to 0.5. For CCR, the JRFS and the HRFS algorithm has the best optimization when the CCR is equal to 5, which is shown in Fig. 5(f).

4.2. Simulations for the real DAG application

4.2.1. Fast Fourier transformation

A fast Fourier transform (FFT) is an algorithm for calculating the discrete fourier transform and its inverse transformation. The input vector size of FFT in Fig. 6(a) is 4. It includes two parts: recursive call tasks (above the dashed line) and the butterfly operation tasks (ones below the line).

For the FFT-related experiments, only the CCR and range percentage parameters were used. Fig. 6 shows the results for FFT graphs of six different CCR values when the number of processors is set as 16. As can be observed from these two figures, the JRFS slightly outperforms the HRFS algorithm in all cases. In the Fig. 6(c), we can observe that the performance improvement of the HRFS algorithm compared with HEFT with a lookahead for FFT graphs gradually increase with the increase of CCR, while the performance improvement for JRFS reaches the maximum value when the CCR is 5.

4.2.2. Gaussian elimination

Gaussian elimination is an algorithm for solving systems of linear equations in mathematics, which is also regarded as a sequence of operations performed on the associated matrix of coefficients. For the experiments of gaussian elimination application, the same CCR and number of processors and range percentage values were used. Since the structure of the application graph is known, parallelism factor does not need to be considered. Matrix size s is used to describe DAG size S, S represents the number of tasks in the DAG graph. The total number of tasks in a gaussian elimination graph is equal to $S = \frac{1}{2}(s^2 + s - 2)$. Fig. 7(a) shows a gaussian elimination graph with matrix size 5. Each $T_{k,k}$ represents a pivot column operation and each $T_{k,j}$ represents an update operation.

Fig. 7 (b) gives the average makespan values of two proposed algorithms at various matrix sizes from 5 to 20, with an increment of one, when the number of processors is equal to 16 and CCR is equal to 1. As the size of a matrix varies from 5 to 20 with an increment step by 1, the total number of task nodes ranges from 14 to 209. It can seen that JRFS performs slightly better than HRFS for gaussian elimination. In Fig. 7(c), we can observe that the performance improvement of the two proposed algorithms compared with HEFT with a lookahead for gaussian elimination gradually increase to a maximum and then gradually decrease with the increase of DAG size.

5. Conclusions

In this paper, we focus on task sequence offloading under task dependency constraints and computing frequency adjusting in MEC. The aim of our work is to find an optimized scheduling mechanism such that makespan is minimized. In our work, the EFT minimum principle is used to impact the mapping relationship between tasks and servers by adjusting the sequence offloading order, and the DVFS technology is adopted to scale frequencies of servers in the MEC center. The results show the capability of the proposed algorithm in achieving a smaller makespan compared to several other heuristics. In the future work, we will committed to the development and research of PSO algorithm and adopt it to solve MEC problems, and other application research [41,42] will be studied in the future.

Declaration of Competing Interest

We wish to confirm that there are no known conflicts of interest associated with this manuscript.

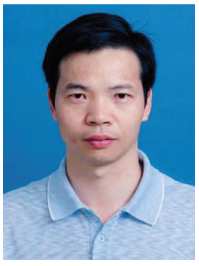
Acknowledgments

The research was partially funded by the National Key R&D Program of China (Grant No. 2018YFB1003401) and the National Natural Science Foundation of China (Grant No. 61702170).

References

[1] F. Wang, J. Xu, X. Wang, S. Cui, Joint offloading and computing optimization in wireless powered mobile-edge computing systems, IEEE Trans. Wirel. Commun. PP (99) (2017) 1.
[2] Y. Mao, J. Zhang, S.H. Song, K.B. Letaief, Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems, IEEE Trans. Wirel. Commun. 16 (9) (2017) 5994–6009.
[3] M. Chiang, T. Zhang, Fog and IoT: An overview of research opportunities, IEEE Int. Things J. 3 (6) (2017) 854–864.
[4] J. Cohen, Embedded speech recognition applications in mobile phones: Status, trends, and challenges, in: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, 2008, pp. 5352–5355.
[5] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, W. Heinzelman, Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture, in: Proceedings of the Computers and Communications, 2012, pp. 000059–000066.
[6] S. Barbarossa, S. Sardellitti, P.D. Lorenzo, Communicating while computing: Distributed mobile cloud computing over 5g heterogeneous networks, IEEE Signal Process. Mag. 31 (6) (2014) 45–55.
[7] P. Bellavista, L. Foschini, D. Scotece, Converging mobile edge computing, fog computing, and IoT quality requirements, in: Proceedings of the IEEE International Conference on Future Internet of Things and Cloud, 2017, pp. 313–320.
[8] A. Ahmed, E. Ahmed, A survey on mobile edge computing, in: Proceedings of the International Conference on Intelligent Systems and Control, 2016.
[9] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, F. Giust, Mobile-edge computing architecture: The role of MEC in the internet of things, IEEE Consum. Electron. Mag. 5 (4) (2016) 84–91.
[10] M.T. Beck, M. Werner, S. Feld, T. Schimper, Mobile edge computing: A taxonomy, in: Proceedings of the the Sixth International Conference on Advances in Future Internet, 2014, pp. 48–54.
[11] T.X. Tran, A. Hajisami, P. Pandey, D. Pompili, Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges, IEEE Commun. Mag. 55 (4) (2017) 54–61.
[12] Y. Yu, J. Zhang, K.B. Letaief, Joint subcarrier and cpu time allocation for mobile edge computing, 2016, pp. 1–6.
[13] S.E. Mahmoodi, R.N. Uma, K.P. Subbalakshmi, Optimal joint scheduling and cloud offloading for mobile applications, IEEE Trans. Cloud Comput. PP (99) (2016) 1.
[14] L. Yang, J. Cao, H. Cheng, Y. Ji, Multi-user computation partitioning for latency sensitive mobile cloud applications, IEEE Trans. Comput. 64 (8) (2015) 2253–2266.
[15] Y. Wang, M. Sheng, X. Wang, L. Wang, J. Li, Mobile-edge computing: Partial computation offloading using dynamic voltage scaling, IEEE Trans. Commun. 64 (10) (2016) 4268–4282.
[16] W. Zhang, Y. Wen, K. Guan, K. Dan, H. Luo, D.O. Wu, Energy-optimal mobile cloud computing under stochastic wireless channel, IEEE Trans. Wirel.s Commun. 12 (9) (2013) 4569–4581.
[17] B.P. Rimal, D.P. Van, M. Maier, Cloudlet enhanced fiber-wireless access networks for mobile-edge computing, IEEE Trans. Wirel. Commun. PP (99) (2017) 1.
[18] Y.H. Kao, B. Krishnamachari, M.R. Ra, B. Fan, Hermes: Latency optimal task assignment for resource-constrained mobile computing, in: Proceedings of the IEEE Conference on Computer Communications, 2015, pp. 1894–1902.
[19] M.R. Ra, A. Sheth, L.B. Mummert, P. Pillai, D. Wetherall, R. Govindan, Odessa: enabling interactive perception applications on mobile devices, in: Proceedings of the International Conference on Mobile Systems, Applications, and Services, 2011, pp. 43–56.
[20] M. Jia, J. Cao, L. Yang, Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing, in: Proceedings of the Computer Communications Workshops, 2014, pp. 352–357.
[21] S. Guo, B. Xiao, Y. Yang, Y. Yang, Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing, in: Proceedings of the IEEE INFOCOM 2016 – the IEEE International Conference on Computer Communications, 2016, pp. 1–9.
[22] K. Kumar, J. Liu, Y.H. Lu, B. Bhargava, A survey of computation offloading for mobile systems, Mobile Netw. Appl. 18 (1) (2013) 129–140.
[23] ETSI, S. Antipolis, France, Mobile-edge computing-introductory technical white paper, Sep. 2014. [Online]. Available: https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_introductory_technical_white_paper_v1%2018-09-14.pdf.
[24] Intel, S. Clara, CA, USA, Real-world impact of mobile edge computing (MEC), Jan. 2016. [Online]. Available: https://builders.intel.com/docs/networkbuilders/Real-world-impact-of-mobile-edgecomputing-MEC.pdf.
[25] M. Satyanarayanan, P. Bahl, R. Cceres, N. Davies, The case for VM-based cloudlets in mobile computing, IEEE Pervas. Comput. 8 (4) (2009) 14–23.
[26] K. Kumar, Y.H. Lu, Cloud computing for mobile users: Can offloading computation save energy? Computer 43 (4) (2010) 51–56.
[27] S.E. Mahmoodi, K.P. Subbalakshmi, V. Sagar, Cloud offloading for multi-radio enabled mobile devices[C], IEEE international conference on communications (ICC), IEEE, 2015, pp. 5473–5478.
[28] S.T. Hong, H. Kim, Qoe-aware computation offloading scheduling to capture energy-latency tradeoff in mobile clouds, in: Proceedings of the IEEE International Conference on Sensing, Communication, and NETWORKING, 2016, pp. 1–9.
[29] Y. Mao, C. You, J. Zhang, et al., Mobile edge computing: Survey and research outlook[J], 2017 arXiv preprint arXiv:1701.01090.
[30] Y. Mao, J. Zhang, K.B. Letaief, Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems, in: Proceedings of the Wireless Communications and NETWORKING Conference, 2017, pp. 1–6.
[31] Y. Ge, Y. Zhang, Q. Qiu, Y.H. Lu, A game theoretic resource allocation for overall energy minimization in mobile cloud computing system, in: Proceedings of the CM/IEEE International Symposium on Low Power Electronics and Design, 2012, pp. 279–284.
[32] A. Ouyang, K. Li, T.K. Truong, A. Sallam, H.M. Sha, Hybrid particle swarm optimization for parameter estimation of muskingum model, Neural Compu. Appl. 25 (7-8) (2014) 1785–1799.
[33] A. Ouyang, Z. Tang, X. Zhou, Y. Xu, G. Pan, K. Li, Parallel hybrid PSO with cuda for ld heat conduction equation, Comput. Fluids 110 (2015) 198–210.
[34] F. Guo, H. Zhang, H. Ji, X. Li, V.C.M. Leung, An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing, IEEE/ACM Trans. Netw. 26 (6) (2018) 2651–2664.
[35] T.G. Rodrigues, K. Suto, H. Nishiyama, N. Kato, K. Temma, Cloudlets activation scheme for scalable mobile edge computing with transmission power control and virtual machine migration, IEEE Trans. Comput. 67 (9) (2018) 1287–1300.
[36] A.P. Miettinen, J.K. Nurminen, Energy efficiency of mobile clients in cloud computing, in: Proceedings of the Usenix Conference on Hot Topics in Cloud Computing, 2010, p. 4.
[37] C. Xiong, G.Y. Li, S. Zhang, Y. Chen, S. Xu, Energy-efficient resource allocation in OFDMA networks, IEEE Trans. Commun. 60 (12) (2012) 3767–3778.
[38] H. Topcuoglu, S. Hariri, M.Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, IEEE Transactions on Parallel & Distributed Systems 13 (3) (2002) 260–274.
[39] E. Ilavarasan, P. Thambidurai, R. Mahilmannan, Performance effective task scheduling algorithm for heterogeneous computing system, J. Comput. Sci. 3 (2) (2007) 28–38.
[40] L.F. Bittencourt, R. Sakellariou, E.R.M. Madeira, Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm, in: Proceeding of the Euromicro International Conference on Parallel, Distributed and Network-Based Processing, 2010, pp. 27–34.
[41] N. Zeng, H. Zhang, B. Song, et al., Facial expression recognition via learning deep sparse autoencoders [J], Neurocomputing 273 (2018) 643–649.
[42] Z. Nianyin, W. Zidong, Z. Bachar, L. Yurong, D. Min, X. Liang, L. Xiaohui, Y. Terry, Image-based quantitative analysis of gold immunochromatographic strip via cellular neural network approach, IEEE Trans. Med. Imaging 33 (5) (2014) 1129–1136 .

Jie Liang received the B.S. degree in communication engineering from Henan normal university in 2011, and the M.S. degree in information and communication engineering from Hunan university in 2014. She is currently working toward the Ph.D. degree at Hunan University, China. Her research interests are mainly in modeling and scheduling of distributed computing systems, optimization and parallel algorithms, game theory, grid and cloud computing.



Kenli Li received the Ph.D. degree in computer science from Huazhong University of Science and Technology, China, in 2003. He was a visiting scholar with the University of Illinois at Urbana-Champaign from 2004 to 2005. He is currently the dean and a full professor of computer science and technology with Hunan University and deputy director of National Supercomputing Center in Changsha. His major research areas include parallel computing, high-performance computing, grid, and cloud computing. He has published more than 150 research papers in international conferences and journals such as the IEEE Transactions on Computers, the IEEE Transactions on Parallel and Distributed Systems}, the IEEE Transactions on Signal Processing, the Journal of Parallel and Distributed Computing, ICPP, and CCGrid. He serves on the editorial board of the IEEE Transactions on Computers. He is an outstanding member of the CCF. He is a senior member of the IEEE.

Chubo Liu received the B.S. degree and Ph.D. degree in computer science and technology from Hunan University, China, in 2011 and 2016, respectively. He is currently an associate professor of computer science and technology at Hunan University. His research interests are mainly in game theory, approximation and randomized algorithms, cloud and edge computing. He has published over 8 papers in journals and conferences such as the IEEE Transactions on Parallel and Distributed Systems}, the IEEE Transactions on Cloud Computing, the ACM Transactions on Modeling and Performance Evaluation of Computing Systems, the Theoretical Computer Science, and ICPADS.

Keqin Li is a SUNY Distinguished Professor of computer science. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber-physical systems. He has published over 500 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, IEEE Transactions on Cloud Computing, IEEE Transactions on Services Computing, IEEE Transactions on Sustainable Computing.