Scheduling Precedence Constrained Tasks for Mobile Applications in Fog Computing

Keqin Li, *Fellow, IEEE*

Abstract—We consider scheduling precedence constrained tasks of a mobile application in a fog computing environment, which faces multiple challenges of precedence constraints, power allocation, and performance-cost tradeoff. Our strategies to handle the three challenges are described as follows. First, in pre-power-allocation algorithms and post-power-allocation algorithms, precedence constraints are handled by the classic list scheduling algorithm and the level-by-level scheduling method respectively. Second, in a pre-power-allocation algorithm (a post-power-allocation algorithm, respectively), a power allocation strategy is determined before (after, respectively) a computation offloading strategy is decided. Third, the performance-cost tradeoff is dealt with by defining the energy-constrained scheduling problem and the time-constrained scheduling problem. That is, between performance and cost, we fix one and minimize the other. The main contributions of the present paper are highlighted as follows. We develop a class of pre-power-allocation algorithms for both energy-constrained and time-constrained scheduling, which are based on the classic list scheduling algorithm and the equal-energy method. We develop a class of post-power-allocation algorithms for both energy-constrained and time-constrained scheduling, which are based on the level-by-level scheduling method and our previously proposed algorithms for independent tasks. We evaluate the proposed algorithms by extensive experiments on mobile applications with randomly generated directed acyclic graphs and identify the most effective and efficient heuristic algorithms. Our research in this paper studies computation offloading in the context of traditional task scheduling while incorporating new and unique features of fog computing into consideration. To the author's best knowledge, there has been no such and similar study in the current literature.

Index Terms—Energy-constrained scheduling, fog computing, level-by-level scheduling, list scheduling, mobile application, post-power-allocation algorithm, pre-power-allocation algorithm, precedence constrained tasks, task scheduling, time-constrained scheduling

1 Introduction

1.1 Challenges and Motivation

Mobile applications in mobile edge computing, fog computing, embedded systems, and Internet of things (IoT) are more and more powerful and sophisticated, such as connected vehicles, face detection and recognition, healthcare, image processing, intelligent video acceleration, interactive gaming, IoT gateway, mobile Big Data analytics, natural language processing, reality augmentation, smart homes and enterprises, and speech recognition. Typically, a mobile application generated on a *user equipment* (UE) can be decomposed into numerous tasks with precedence constraints which can be arbitrarily complicated [23]. Furthermore, the tasks may have very different computation and communication requirements.

Such a complicated mobile application is beyond the computing capability of a mobile device for timely processing. With the assistance of servers in *mobile edge clouds* (MECs), tasks of a mobile application can be offloaded to the MEC servers. Computation offloading provides an efficacious means to enhance the computing power of a UE and to extend the battery lifetime of a UE. By parallel and possibly faster task execution on the MECs, a UE may complete an application in shorter time, at the cost of extra time for communication. By remote task processing on the MECs, a UE may save energy consumption for computation and make its battery to last longer, at the cost of extra energy for communication.

Computation offloading for a mobile application with precedence constrained tasks becomes scheduling precedence constrained tasks of a mobile application in a fog computing environment. However, fog computing introduces several new and unique features that are quite different from traditional energy-efficient task scheduling systems, and a fog computing environment is a sophisticated and hard-to-manage computing platform. First, a UE does not offload all its tasks to the MECs. In fact, a UE also has task execution capability. In other words, a task may not be offloaded and executed locally on the UE or may be offloaded to an MEC and executed remotely on the MEC. Second, a UE cannot change and control the computation speeds of the MECs, but only its own computation speed and its communication speeds to the MECs. In other words, a UE can only partially determine the execution time of a task. Third, only the energy consumption of computation and communication in the UE is considered in dealing with the energy-delay tradeoff. In other words, energy consumption in the MECs is not included into problem formulation and solution. Fourth, fog computing exhibits strong heterogeneity, that is, a task has different execution times and energy consumptions on the UE and the MECs due to different computation and communication speeds and different characteristics of communication channels.

- *The author is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA. E-mail: lik@newpaltz.edu.*

Manuscript received 15 March 2022; revised 30 June 2022; accepted 15 July 2022. Date of publication 0 2022; date of current version 0 2022.
Digital Object Identifier no. 10.1109/TSC.2022.3192095

There are multiple challenges in scheduling precedence constrained tasks of a mobile application in a heterogeneous fog computing environment. First, a computation offloading strategy needs to be decided, which tells when and where to execute a task, such that all precedence constraints among the tasks are satisfied. Second, a power allocation strategy needs to be determined, which, for each task, gives the computation speed for local execution or the communication speed for remote execution. Third, both performance (i.e., overall execution time) and cost (i.e., total energy consumption) should be included into consideration when an optimization problem is defined. It is already known that task scheduling is NP-hard even for independent tasks and only one MEC, and certainly becomes more challenging with added concerns of precedence constraints, power allocation, and performance-cost tradeoff, and the inability of a UE to change and control the computation speeds of the MECs.

The motivation of this paper is to develop high-quality heuristic algorithms for scheduling precedence constrained tasks of a mobile application in a fog computing environment by effectively handling all the above challenges. Our investigation is conducted within a well developed framework, which can inspire further research and better algorithms.

1.2 New Contributions

In this paper, we consider scheduling precedence constrained tasks of a mobile application in a fog computing environment. Our strategies to handle the three challenges are described as follows. First, in pre-power-allocation algorithms and post-power-allocation algorithms, precedence constraints are handled by the classic list scheduling algorithm and the level-by-level scheduling method respectively. Second, in a *pre-power-allocation algorithm* (a *post-power-allocation algorithm*, respectively), a power allocation strategy is determined before (after, respectively) a computation offloading strategy is decided. Third, the performance-cost tradeoff is dealt with by defining the energy-constrained scheduling problem and the time-constrained scheduling problem. That is, between performance and cost, we fix one and minimize the other. Using the above strategies, scheduling precedence constrained tasks of a mobile application in a fog computing environment can be investigated systematically, and various heuristic algorithms can be developed and their performance can be evaluated and compared.

The main contributions of the present paper are highlighted as follows.

- We develop a class of pre-power-allocation algorithms for both energy-constrained and time-constrained scheduling, which are based on the classic list scheduling algorithm and the equal-energy method.
- We develop a class of post-power-allocation algorithms for both energy-constrained and time-constrained scheduling, which are based on the level-by-level scheduling method and our previously proposed algorithms for independent tasks.
- We evaluate the proposed algorithms by extensive experiments on mobile applications with randomly generated directed acyclic graphs and identify the most effective and efficient heuristic algorithms.

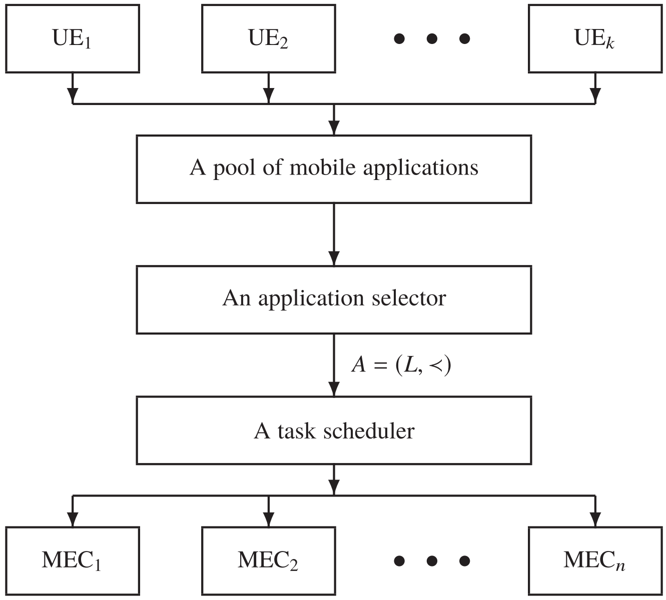

Fig. 1. A service-oriented fog computing environment.

Our research in this paper studies computation offloading in the context of traditional task scheduling while incorporating new and unique features of fog computing into consideration. To the author's best knowledge, there has been no such and similar study in the current literature. However, the techniques of pre-power-allocation, post-power-allocation, list scheduling, level-by-level scheduling, energy-constrained scheduling, and time-constrained scheduling are all borrowed from traditional energy-efficient task scheduling in parallel and distributed computing systems [11], [12].

The organization of the paper is summarized as follows. In Section 2, we provide background information, including the models used in the paper, problem definitions, and NP-hardness. In Section 3, we develop pre-power-allocation algorithms. In Section 4, we develop post-power-allocation algorithms. In Section 5, we experimentally evaluate the performance of our proposed algorithms. In Section 6, we review related research. In Section 7, we conclude the paper.

2 Background Information

In this section, we present the models used in the paper, define our scheduling problems, and show their NP-hardness.

Our task scheduling problem incorporated into a service-oriented fog computing environment is illustrated in Fig. 1. In such an environment, there are multiple UEs, multiple MECs, an application selector facing the UEs, and a task scheduler facing the MECs. The UEs can submit service requests in the form of mobile applications (see Section 2.1) that are put into an application pool. An *application selector* (i.e., a request server) chooses the next application (i.e., service request) to be processed according to certain criterion (e.g., quality of service). Once an application is chosen, a *task scheduler* decides when, where, and how to execute the tasks of the application on the multiple MECs (see Sections 2.2, and 2.3). Note that in this article, we focus on the design of the task scheduler, which includes a computation offloading strategy and a power allocation strategy (see Section 2.4).

2.1 The Application Model

In this section, we describe the mobile application model.

Let us assume that a UE has a *mobile application* $A = (L, \prec)$, which is specified as follows.

There is a list of tasks $L = (t_1, t_2, \ldots, t_m)$. Each task t_i is specified as $t_i = (r_i, d_i)$, where r_i is the computation requirement (i.e., the amount of computation, measured by the number of billion processor cycles or the number of billion instructions (BI) to be executed) of t_i, and d_i is the communication requirement (i.e., the amount of data to be communicated between the UE and an MEC, measured by the number of million bits (MB)) of t_i.

There are precedence constraints among the tasks, which are specified by a partial order $\prec$. If $t_{i_1} \prec t_{i_2}$, then t_{i_1} is a predecessor of t_{i_2}, and task t_{i_2} cannot start its execution until task t_{i_1} is completed. A mobile application with precedence constrained tasks can be described by a *directed acyclic graph* (dag) G. The vertices in G are the m tasks in L. The arcs in G are given in such a way that there is an arc from t_{i_1} to t_{i_2} if and only if $t_{i_1} \prec t_{i_2}$.

2.2 The Computation and Communication Models

In this section, we describe the task execution model.

Assume that there are n heterogeneous MECs, i.e., MEC_1, MEC_2,..., MEC_n. Each MEC_j has computation speed s_j (i.e., the processor execution speed, measured by GHz or the number of billion instructions that can be executed in one second), which cannot be changed by the UE, for all $1 \le j \le n$.

Each task t_i can be executed on the UE or an MEC. Task execution time includes computation time and communication time.

If t_i is not offloaded and executed locally on the UE with computation speed $s_{0,i}$, which can be decided by the UE, the computation time (measured by seconds) of t_i on the UE is $r_i/s_{0,i}$. There is no communication time for local execution. The execution time of t_i with local execution on the UE is $T_i = r_i/s_{0,i}$, for all $1 \le i \le m$.

If t_i is offloaded to an MEC_{j_i} and executed remotely on MEC_{j_i}, the computation time of t_i on MEC_{j_i} is r_i/s_{j_i}. The communication speed between the UE and MEC_{j_i} for t_i is c_{i,j_i} (i.e., the data transmission rate, measured by the number of million bits that can be transmitted in one second), which can be decided by the UE. The communication time (measured by seconds) between the UE and MEC_{j_i} for t_i is $d_i/c_{i,j_i}$. The execution time of t_i with remote execution on MEC_{j_i} is $T_i = r_i/s_{j_i} + d_i/c_{i,j_i}$, for all $1 \le i \le m$ and $1 \le j_i \le n$.

2.3 The Power Consumption Models

In this section, we describe the power consumption models for both computation and communication.

There are two components in the UE's power consumption P (measured by Watts) for computation, i.e., dynamic power consumption and static power consumption. The dynamic component P_d is typically represented as $P_d = \xi s_0^\alpha$, where ξ and α are some constants determined by the technology. The static component P_s is normally a constant. Consequently, we get $P = P_d + P_s = \xi s_0^\alpha + P_s$. If t_i is not offloaded and executed locally on the UE with computation speed $s_{0,i}$, the power consumption is $P_i = \xi s_{0,i}^\alpha + P_s$, and the energy consumption for computation (measured by Joules) of t_i on the UE is $E_i = P_i(r_i/s_{0,i}) = ((\xi s_{0,i}^\alpha + P_s)/s_{0,i})r_i$, for all $1 \le i \le m$.

Note that a UE consumes power for communication in addition to consuming power for computation. Let P_{t,i,j_i} be the transmission power (measured by Watts) of the UE to MEC_{j_i} for task t_i, where $1 \le i \le m$ and $1 \le j_i \le n$. Then, we have $P_{t,i,j_i} = (2^{c_{i,j_i}/w_{j_i}} - 1)/\beta_{j_i}$, for all $1 \le i \le m$ and $1 \le j_i \le n$, where w_{j_i} is the channel bandwidth and β_{j_i} is a quantity combining various factors such as the background noise power, the interference on the communication channel caused by other devices' data transmission to the same MEC, and the channel gain between the UE and MEC_{j_i}. The energy consumption for communication (measured by Joules) of t_i from the UE to MEC_{j_i} is $E_i = P_{t,i,j_i}(d_i/c_{i,j_i}) = (2^{c_{i,j_i}/w_{j_i}} - 1)/(\beta_{j_i} c_{i,j_i})d_i$, for all $1 \le i \le m$ and $1 \le j_i \le n$.

Notice that for local execution on the UE, only energy consumption for computation is considered, and for remote execution on an MEC, only energy consumption for communication is considered. The total energy consumption of a mobile application is $E = \sum_{i=1}^m E_i$, which is the main cost measure of a mobile application.

2.4 Problem Definitions

In this section, we formally define our optimization problems to be solved in this paper.

A *computation offloading strategy* (a.k.a. *schedule*) of a mobile application A is to decide for each task t_i, *when* (the starting time τ_i of execution) and *where* (the location, either the UE or an MEC, of execution) to execute t_i, where $1 \le i \le m$. A legitimate schedule must ensure that all tasks follow the precedence constraints, i.e., $\tau_{i_1} + T_{i_1} \le \tau_{i_2}$, if $t_{i_1} \prec t_{i_2}$. A *power allocation strategy* is to decide for each task t_i, *how* (the computation speed $s_{0,i}$ for local execution on the UE or the communication speed c_{i,j_i} for remote execution on MEC_{j_i}) to execute t_i, where $1 \le i \le m$.

We use T to denote the overall execution time to finish all the tasks in L (i.e., the *makespan*), which is the main performance measure of a mobile application.

Given a mobile application $A = (L, \prec)$ of a UE, where $L = (t_1, t_2, \ldots, t_m)$, with $t_i = (r_i, d_i)$, for all $1 \le i \le m$, in a fog computing environment with n MECs, i.e., MEC_1, MEC_2,..., MEC_n, where MEC_j has computation speed s_j, for all $1 \le j \le n$, and an energy constraint $\tilde{E}$, the *energy-constrained scheduling problem* is to find a computation offloading strategy and a power allocation strategy for all tasks in L on the UE and MECs, such that E does not exceed $\tilde{E}$ and T is minimized.

Given a mobile application $A = (L, \prec)$ of a UE, where $L = (t_1, t_2, \ldots, t_m)$, with $t_i = (r_i, d_i)$, for all $1 \le i \le m$, in a fog computing environment with n MECs, i.e., MEC_1, MEC_2,..., MEC_n, where MEC_j has computation speed s_j, for all $1 \le j \le n$, and a time constraint $\tilde{T}$, the *time-constrained scheduling problem* is to find a computation offloading strategy and a power allocation strategy for all tasks in L on the UE and MECs, such that T does not exceed $\tilde{T}$ and E is minimized.

2.5 NP-Hardness

In this section, we show that even for very special cases, e.g., for independent tasks and only one MEC, our combinatorial optimization problems are still NP-hard.

Theorem 1. *The energy-constrained scheduling problem is NP-hard even for independent tasks and only one MEC.*

Proof. Assume that tasks $t_1, t_2, \ldots, t_{m'}$ are executed on the UE with total energy consumption $\tilde{E}$. We can show that the overall execution time T on the UE is minimized when all the tasks have the same computation speed on the UE. Let us assume that t_i is executed with computation speed $s_{0,i}$ on the UE, where $1 \le i \le m'$. Then, we have

$$T(s_{0,1}, s_{0,2}, \ldots, s_{0,m'}) = \sum_{i=1}^{m'} r_i / s_{0,i}, \tag{1}$$

and

$$E(s_{0,1}, s_{0,2}, \ldots, s_{0,m'}) = \sum_{i=1}^{m'} (\xi s_{0,i}^{\alpha-1} + P_s / s_{0,i}) r_i, \tag{2}$$

where both the overall execution time $T(s_{0,1}, s_{0,2}, \ldots, s_{0,m'})$ and the total energy consumption $E(s_{0,1}, s_{0,2}, \ldots, s_{0,m'})$ are viewed as functions of the computation speeds $s_{0,1}, s_{0,2}, \ldots, s_{0,m'}$. To minimize $T(s_{0,1}, s_{0,2}, \ldots, s_{0,m'})$ subject to the constraint $E(s_{0,1}, s_{0,2}, \ldots, s_{0,m'}) = \tilde{E}$, we use the Lagrange multiplier system

$$\nabla T(s_{0,1}, s_{0,2}, \ldots, s_{0,m'}) = \lambda \nabla E(s_{0,1}, s_{0,2}, \ldots, s_{0,m'}), \tag{3}$$

where λ is a Lagrange multiplier. Since

$$\frac{\partial T(s_{0,1}, s_{0,2}, \ldots, s_{0,m'})}{\partial s_i} = \lambda \frac{\partial E(s_{0,1}, s_{0,2}, \ldots, s_{0,m'})}{\partial s_i}, \tag{4}$$

that is

$$-\frac{r_i}{s_{0,i}^2} = \lambda r_i \left(\xi(\alpha-1) s_{0,i}^{\alpha-2} - \frac{P_s}{s_{0,i}^2} \right), \tag{5}$$

we have

$$s_{0,i} = s_0 = \left(\frac{1}{\xi(\alpha-1)} \left(P_s - \frac{1}{\lambda} \right) \right)^{1/\alpha}, \tag{6}$$

for all $1 \le i \le m'$. Substituting the above $s_{i,0}$ into the constraint $E(s_{0,1}, s_{0,2}, \ldots, s_{0,m'}) = \tilde{E}$, we get

$$R\left(\xi s_0^{\alpha-1} + \frac{P_s}{s_0} \right) = \tilde{E}, \tag{7}$$

where $R = r_1 + r_2 + \cdots + r_{m'}$ is the total execution requirement of the m' tasks. The above discussion implies that the overall execution time T on the UE is minimized when all the tasks have the same computation speed s_0 on the UE, which can be found by solving the equation

$$\xi s_0^{\alpha} - (\tilde{E}/R) s_0 + P_s = 0. \tag{8}$$

For instance, when $\alpha = 2$, we have

$$s_0 = \frac{1}{2\xi} \left(\tilde{E}/R + \sqrt{(\tilde{E}/R)^2 - 4\xi P_s} \right). \tag{9}$$

Assume that tasks $t_1, t_2, \ldots, t_{m'}$ are executed on MEC_1 with total energy consumption $\tilde{E}$. We can show that the overall execution time T on the MEC is minimized when all the tasks have the same communication speed between the UE and MEC_1. Let us assume that t_i is executed with communication speed $c_{i,1}$ between the UE and MEC_1, where $1 \le i \le m'$. Then, we have

$$T(c_{1,1}, c_{2,1}, \ldots, c_{m,'1}) = \sum_{i=1}^{m'} (r_i / s_1 + d_i / c_{i,1}), \tag{10}$$

and

$$E(c_{1,1}, c_{2,1}, \ldots, c_{m,'1}) = \sum_{i=1}^{m'} \left(\frac{2^{c_{i,1}/w_1} - 1}{\beta_1 c_{i,1}} \right) d_i, \tag{11}$$

where both the overall execution time $T(c_{1,1}, c_{2,1}, \ldots, c_{m,'1})$ and the total energy consumption $E(c_{1,1}, c_{2,1}, \ldots, c_{m,'1})$ are viewed as functions of the communication speeds $c_{1,1}, c_{2,1}, \ldots, c_{m,'1}$. To minimize $T(c_{1,1}, c_{2,1}, \ldots, c_{m,'1})$ subject to the constraint $E(c_{1,1}, c_{2,1}, \ldots, c_{m,'1}) = \tilde{E}$, we use the Lagrange multiplier system

$$\nabla T(c_{1,1}, c_{2,1}, \ldots, c_{m,'1}) = \lambda \nabla E(c_{1,1}, c_{2,1}, \ldots, c_{m,'1}), \tag{12}$$

where λ is a Lagrange multiplier. Since

$$\frac{\partial T(c_{1,1}, c_{2,1}, \ldots, c_{m,'1})}{\partial c_{i,1}} = \lambda \frac{\partial E(c_{1,1}, c_{2,1}, \ldots, c_{m,'1})}{\partial c_{i,1}}, \tag{13}$$

that is

$$-\frac{d_i}{c_{i,1}^2} = \lambda d_i \left(\frac{2^{c_{i,1}/w_1} (\ln 2 / w_1) c_{i,1} - (2^{c_{i,1}/w_1} - 1)}{\beta_1 c_{i,1}^2} \right), \tag{14}$$

we have $c_{i,1} = c_1$ for all $1 \le i \le m'$. Substituting the above $c_{i,1}$ into the constraint $E(c_{1,1}, c_{2,1}, \ldots, c_{m,'1}) = \tilde{E}$, we get

$$\left(\frac{2^{c_1/w_1} - 1}{\beta_1 c_1} \right) D = \tilde{E}, \tag{15}$$

where $D = d_1 + d_2 + \cdots + d_{m'}$ is the total communication requirement of the m' tasks. The above discussion implies that the overall execution time T on the MEC is minimized when all the tasks have the same communication speed c_1 between the UE and MEC_1, which can be found by solving the equation

$$2^{c_1/w_1} - (\tilde{E}/D) \beta_1 c_1 - 1 = 0. \tag{16}$$

When tasks have the same computation speed on the UE and the same communication speed between the UE and MEC_1, the energy-constrained scheduling problem becomes the *problem of optimal computation offloading with energy constraint* [13], which has been proven to be NP-hard for independent tasks and only one MEC. □

Theorem 2. *The time-constrained scheduling problem is NP-hard even for independent tasks and only one MEC.*

Proof. The proof follows a similar argument to that of the proof of Theorem 1.

Assume that tasks $t_1, t_2, \ldots, t_{m'}$ are executed on the UE with overall execution time $\tilde{T}$. We can show that total energy consumption E for computation is minimized when all the tasks have the same computation speed on the UE.

Assume that tasks $t_1, t_2, \ldots, t_{m'}$ are executed on MEC$_1$ with overall execution time $\tilde{T}$. We can show that total energy consumption E for communication is minimized when all the tasks have the same communication speed between the UE and MEC$_1$.

When tasks have the same computation speed on the UE and the same communication speed between the UE and MEC$_1$, the time-constrained scheduling problem becomes the *problem of optimal computation offloading with time constraint* [13], which has been proven to be NP-hard for independent tasks and only one MEC. □

The remaining of the paper is to seek heuristic algorithms which are able to produce high-quality solutions.

3 Pre-Power-Allocation Algorithms

In this section, we develop pre-power-allocation algorithms. In these algorithms, a power allocation strategy is determined before a computation offloading strategy is decided.

3.1 Energy-Constrained Scheduling

In this section, we consider energy-constrained scheduling with pre-power-allocation.

There are several methods for pre-power-allocation [11]. In the *equal-speed* method, all tasks have the same computation speed. This is not possible in fog computing, since the UE cannot change the computation speed of an MEC. In the *equal-time* method, all tasks have the same execution time. This is again not possible in fog computing, since the UE can only control the communication time. In this paper, we adopt the *equal-energy* method, in which, all tasks consume the same amount of energy, i.e., $\tilde{E}/m$. The advantage is that when a task is scheduled on the UE or an MEC, its computation or communication speed can be decided immediately.

If t_i is not offloaded and executed locally on the UE with computation speed $s_{0,i}$, we have

$$E_i = (\xi s_{0,i}^{\alpha-1} + P_s/s_{0,i})r_i = \tilde{E}/m, \tag{17}$$

that is

$$\xi s_{0,i}^{\alpha} - (\tilde{E}/(mr_i))s_{0,i} + P_s = 0. \tag{18}$$

When $\alpha = 2$, we get

$$s_{0,i} = \frac{1}{2\xi}\left(\tilde{E}/(mr_i) + \sqrt{(\tilde{E}/(mr_i))^2 - 4\xi P_s}\right). \tag{19}$$

In general, we observe that $\xi s_{0,i}^{\alpha} - (\tilde{E}/(mr_i))s_{0,i} < 0$, which implies that $s_{0,i} < (\tilde{E}/(\xi mr_i))^{1/(\alpha-1)}$. Hence, Eq. (18) can be solved numerically by using the standard bisection method, which searches for $s_{0,i}$ in the interval $[0, (\tilde{E}/(\xi mr_i))^{1/(\alpha-1)}]$. However, as mentioned in [13]

$$E_i \ge r_i P_s^{1-1/\alpha}\xi^{1/\alpha}\frac{\alpha}{(\alpha-1)^{1-1/\alpha}}. \tag{20}$$

Thus, Eq. (18) has a solution only if

$$\tilde{E} \ge mr_i P_s^{1-1/\alpha}\xi^{1/\alpha}\frac{\alpha}{(\alpha-1)^{1-1/\alpha}}. \tag{21}$$

For instance, when $\alpha = 2$, we must have

$$\tilde{E} \ge 2mr_i\sqrt{\xi P_s}. \tag{22}$$

If t_i is offloaded to an MEC$_{j_i}$ and executed remotely on MEC$_{j_i}$, we have

$$E_i = \left(\frac{2^{c_{i,j_i}/w_{j_i}} - 1}{\beta_{j_i} c_{i,j_i}}\right)d_i = \tilde{E}/m, \tag{23}$$

that is

$$2^{c_{i,j_i}/w_{j_i}} - (\tilde{E}/(md_i))\beta_{j_i} c_{i,j_i} - 1 = 0. \tag{24}$$

By using a Taylor series, we know that for an exponential function b^x, we have

$$b^x > 1 + (\ln b)x + \frac{1}{2}(\ln b)^2 x^2, \tag{25}$$

where we notice that $(b^x)' = b^x \ln b$ and $(b^x)'' = b^x(\ln b)^2$. By letting $b = 2^{1/w_{j_i}}$ and $x = c_{i,j_i}$, we get

$$2^{c_{i,j_i}/w_{j_i}} > 1 + (\ln 2/w_{j_i})c_{i,j_i} + \frac{1}{2}(\ln 2/w_{j_i})^2 c_{i,j_i}^2, \tag{26}$$

and

$$(\ln 2/w_{j_i})c_{i,j_i} + \frac{1}{2}(\ln 2/w_{j_i})^2 c_{i,j_i}^2 < (\tilde{E}/(md_i))\beta_{j_i} c_{i,j_i}, \tag{27}$$

which implies that

$$c_{i,j_i} < c_{i,j_i}^* = \frac{2((\tilde{E}/(md_i))\beta_{j_i} - (\ln 2/w_{j_i}))}{(\ln 2/w_{j_i})^2}. \tag{28}$$

Hence, Eq. (24) can be solved numerically by using the standard bisection method, which searches for c_{i,j_i} in the interval $[0, c_{i,j_i}^*]$. As mentioned in [13]

$$E_i > \left(\frac{\ln 2}{w_{j_i}\beta_{j_i}}\right)d_i. \tag{29}$$

Thus, Eq. (24) has a solution only if

$$\tilde{E} > m\left(\frac{\ln 2}{w_{j_i}\beta_{j_i}}\right)d_i. \tag{30}$$

Our energy-constrained scheduling algorithm with pre-power-allocation, called Energy-Constrained List Scheduling with Heuristic H (ECLS-H), is presented in Algorithm 1 (see Section 5.1 for H).

Notation: In this paper, we define

$$\text{indexmin}(x_1, x_2, \ldots, x_n),$$

to be the index j such that $x_j = \min(x_1, x_2, \ldots, x_n)$. Similarly, we define

$$\text{indexmax}(x_1, x_2, \ldots, x_n),$$

to be the index j such that $x_j = \max(x_1, x_2, \ldots, x_n)$.

The algorithm is essentially the classic *list scheduling* algorithm [3] adapted for a fog computing environment. With pre-power-allocation, the execution time of a task can be available when the task is scheduled for execution.

Algorithm 1. Energy-Constrained List Scheduling With Heuristic H (ECLS-H)

Input: $A = (L, \prec)$ with $L = (t_1, t_2, \ldots, t_m)$, where $t_i = (r_i, d_i)$, for all $1 \le i \le m$, UE $= (\xi, \alpha, P_s)$, MEC$_j = (s_j, w_j, \beta_j)$, for all $1 \le j \le n$, and $\tilde{E}$.
Output: A computation offloading strategy and a power allocation strategy such that E does not exceed $\tilde{E}$ and T is minimized.

Initialize the list L using heuristic H; (1)
$T \leftarrow 0$; (2)
for (each unscheduled ready task t_i) **do** (3)
  if (there is an available MEC$_j$) **then** (4)
    Schedule t_i on MEC$_j$ at time 0; (5)
    $W_j \leftarrow$ the execution time of t_i; (6)
    Remove t_i from L; (7)
  end if; (8)
end do; (9)
while (there is still a running task) **do** (10)
  $j \leftarrow \mathrm{indexmin}_{0 \le j' \le n, W_{j'} \neq 0}(W_{j'})$; (11)
  $T \leftarrow T + W_j$; (12)
  for ($j' = 0$; $j' \le n$; j'++) **do** (13)
    if ($W_{j'} \neq 0$) **then** (14)
      $W_{j'} \leftarrow W_{j'} - W_j$; (15)
    end if; (16)
  end do; (17)
  for (each unscheduled ready task t_i) **do** (18)
    if (there is an available MEC$_j$) **then** (19)
      Schedule t_i on MEC$_j$ at time T; (20)
      $W_j \leftarrow$ the execution time of t_i; (21)
      Remove t_i from L; (22)
    end if; (23)
  end do; (24)
end do. (25)

The list L is initialized with heuristic H (line (1)). The variable T dynamically records the current time as a schedule move on (line (2)). The for-loop in lines (3)–(9) schedules the first batch of ready tasks (line (3)) at time 0 (line (5)). Let W_j (line (6)) denote the remaining execution time of the task currently running on MEC $_j$, for all $0 \le j \le n$, where we set UE $=$ MEC$_0$ for convenience. The while-loop in lines (10)–(25) schedules the remaining tasks. In each repetition, the following actions are performed. First, the MEC$_j$ which completes its current task the earliest is identified (line (11)). Second, the time clock moves on to the moment when MEC$_j$ completes its current task (line (12)). Third, the remaining execution time of each busy MEC (line (14)) is updated (line (15)) by the for-loop in lines (13)–(17). Fourth, the next batch of ready tasks (line (18)) are scheduled at time T (line (20)) by the for-loop in lines (18)–(24). The execution time of t_i (lines (6) and (21)) is $r_i/s_{0,i}$ if $j = 0$, where $s_{0,i}$ is found by solving Eq. (18), and $r_i/s_{j_i} + d_i/c_{i,j_i}$ if $j > 0$, where c_{i,j_i} is found by solving Eq. (24). The algorithm tells when and where (lines (5) and (20)), and how (lines (6) and (21)) to execute t_i, for all $1 \le i \le m$.

When Eqs. (18) or (24) cannot be solved due to insufficient energy allocation, the UE or MEC$_j$ is considered not available and skipped.

The time complexity of the algorithm is analyzed as follows. Line (1) typically takes $O(m \log m)$ time. The for-loop in lines (3)–(9) repeats m times. Line (6) needs to solve Eq. (18) or Eq. (24), which requires $O(\log(I/\epsilon))$ time, where I is the length of the largest search internal in this paper. However, line (6) is performed at most n times. Thus, the for-loop in lines (3)–(9) takes $O(m + n\log(I/\epsilon))$ time. The while-loop in lines (10)–(25) repeats m times, one for each completed task. In each repetition of the while-loop, line (11) requires $O(n)$ time; the for-loop in lines (13)–(17) requires $O(n)$ time; the for-loop in lines (18)–(24) requires $O(m + n\log(I/\epsilon))$ time. Therefore, the while-loop in lines (10)–(25), and the overall time complexity of Algorithm 1 is $O(m(m + n\log(I/\epsilon)))$.

3.2 Time-Constrained Scheduling

In this section, we consider time-constrained scheduling with pre-power-allocation.

Our time-constrained scheduling algorithm with pre-power-allocation, called Time-Constrained List Scheduling with Heuristic H (TCLS-H), is presented in Algorithm 2.

Algorithm 2. Time-Constrained List Scheduling With Heuristic H (TCLS-H)

Input: $A = (L, \prec)$ with $L = (t_1, t_2, \ldots, t_m)$, where $t_i = (r_i, d_i)$, for all $1 \le i \le m$, UE $= (\xi, \alpha, P_s)$, MEC$_j = (s_j, w_j, \beta_j)$, for all $1 \le j \le n$, and $\tilde{T}$.
Output: A computation offloading strategy and a power allocation strategy such that T does not exceed $\tilde{T}$ and E is minimized.

$\tilde{E} \leftarrow$ a reasonable value; (1)
do (2)
  Call Algorithm ECLS-H with $\tilde{E}$ to get T; (3)
  $\tilde{E} \leftarrow \tilde{E} + \Delta E$; (4)
while $(T > \tilde{T})$; (5)
$\phi \leftarrow \tilde{T}/T$; (6)
for ($i = 1$; $i \le m$; i++) **do** (7)
  if ($j_i = 0$) **then** (8)
    $s_{0,i} \leftarrow s_{0,i}/\phi$; (9)
  else (10)
    $c_{i,j_i} \leftarrow d_i/(\phi(r_i/s_{j_i} + d_i/c_{i,j_i}) - r_i/s_{j_i})$; (11)
  end if; (12)
end do. (13)

It is very difficult to decide the computation or communication speed when a task is scheduled in time-constrained scheduling to guarantee a time constraint. Our strategy is to adopt a two stage process. In the first stage (lines (1)–(5)), we find $\tilde{E}$ such that T obtained by the ECLS-H algorithm (line (3)) is no longer than $\tilde{T}$ (line (5)). This can be realized by setting $\tilde{E}$ to some reasonable value (line (1)), and gradually increasing $\tilde{E}$ (line (4)) until $T \le \tilde{T}$. In the second stage (lines (6)–(13)), the execution time of each task (line (7)) is scaled by a factor of $\phi = \tilde{T}/T \ge 1$ (line (6)) by reducing the computation or communication speed as follows. If task t_i is scheduled on the UE (line (8)), $s_{0,i}$ is changed to $s'_{0,i}$, such that $r_i/s'_{0,i} = \phi(r_i/s_{0,i})$, which gives $s'_{0,i} = s_{0,i}/\phi$ (line (9)). If task t_i is scheduled on MEC$_j$ (line (10)), c_{i,j_i} is changed to c'_{i,j_i}, such that $r_i/s_{j_i} + d_i/c'_{i,j_i} = \phi(r_i/s_{j_i} + d_i/c_{i,j_i})$, which gives $c'_{i,j_i} = d_i/(\phi(r_i/s_{j_i} + d_i/c_{i,j_i}) - r_i/s_{j_i})$ (line (11)). Notice that such execution time scaling does not affect the the precedence constraints among the tasks and the locations to execute the tasks, only the starting times for execution of the tasks.

After computation and communication speed reduction, tasks no longer consume the same amount of energy. However, this is not an issue at all, since our original purpose is to produced a computation offloading strategy and a power allocation strategy such that the time constraint is satisfied.

It is clear that if the ECLS-H algorithm is called K times in the first stage, the overall time complexity of Algorithm 2 is $O(Km(m + n\log(I/\epsilon)))$. The value K depends on the initial value of $\tilde{E}$ and the increment ΔE.

4 Post-Power-Allocation Algorithms

In this section, we develop post-power-allocation algorithms. In these algorithms, a power allocation strategy is determined after a computation offloading strategy is decided.

4.1 Energy-Constrained Scheduling

In this section, we consider energy-constrained scheduling with post-power-allocation.

A directed acyclic graph can be decomposed into v levels, where the levels are defined as follows. Level 1 consists of initial tasks, i.e., tasks with no predecessors. Generally, level l contains a task t_i if the number of nodes on the longest path from some initial task to task t_i is l, where $1 \le l \le v$. Let L_l denote the set of tasks in level l, for all $1 \le l \le v$. Thus, we have $L = L_1 \cup L_2 \cup \cdots \cup L_v$.

We adopt the *level-by-level scheduling* method, i.e., tasks in L are scheduled level by level. This means that only when all tasks in L_{l-1} are completed, can tasks in L_l start their execution. The schedule of each level is produced individually, independently, and separately. The schedule of the entire mobile application is simply a concatenation of the v schedules for $L_1, L_2, \ldots, L_v$.

Since all tasks in the same level are independent of each other, we can schedule them by using any heuristic energy-constrained scheduling algorithm H for independent tasks, e.g., those developed in [13]. All these algorithms have a unique feature, i.e., a power allocation strategy is determined after a computation offloading strategy is decided.

Our energy-constrained scheduling algorithm with post-power-allocation, called Energy-Constrained Level-by-Level Scheduling with Heuristic H (ECLL-H), is presented in Algorithm 3.

The key issue in level-by-level energy-constrained scheduling is to determine how the given energy budget $\tilde{E}$ is allocated to the v levels. Let $H(L_l, E_l)$ be the overall execution time when algorithm H is applied to L_l with energy constraint E_l. Initially, each level L_l is scheduled by using algorithm H with some initial energy allocation E_l (lines (1)–(3)). Then, the remaining energy $\tilde{E} - (E_1 + E_2 + \cdots + E_v)$ (line (4)) is divided by K to get E' (line (5)), and the while-loop in lines (6)–(16) is repeated slightly more than K times. In each repetition, the following actions are performed. First, ΔE is determined, which is a random number γ times E', where γ is uniformly distributed in [0.5,1.0] (line (10)). Second, the level l' which results in the largest reduction in its overall execution time if ΔE extra energy is provided, i.e., $H(L_l, E_l) - H(L_l, E_l + \Delta E)$, is selected (line (12)). Third, level l' is allocated ΔE extra energy (line (13)). The while-loop terminates after all the remaining energy is allocated (line (6)). The overall execution time T of the mobile application is simply $H(L_1, E_1) + H(L_2, E_2) + \cdots + H(L_v, E_v)$ (line (17)).

Algorithm 3. Energy-Constrained Level-by-Level Scheduling With Heuristic H (ECLL-H)

Input: $A = (L, \prec)$ with $L = (t_1, t_2, \ldots, t_m)$, where $t_i = (r_i, d_i)$, for all $1 \le i \le m$, UE $= (\xi, \alpha, P_s)$, MEC$_j = (s_j, w_j, \beta_j)$, for all $1 \le j \le n$, and $\tilde{E}$.
Output: A computation offloading strategy and a power allocation strategy such that E does not exceed $\tilde{E}$ and T is minimized.

for ($l = 1$; $l \le v$; l++) **do** (1)
  $T_l \leftarrow H(L_l, E_l)$; (2)
end do; (3)
$remainingE \leftarrow \tilde{E} - (E_1 + E_2 + \cdots + E_v)$; (4)
$E' \leftarrow (\tilde{E} - (E_1 + E_2 + \cdots + E_v))/K$; (5)
while ($remainingE > 0$) **do** (6)
  if ($remainingE \le E'$) **then** (7)
    $\Delta E \leftarrow remainingE$; (8)
  else (9)
    $\Delta E \leftarrow \gamma E'$, where $\gamma \in [0.5, 1.0]$; (10)
  end if; (11)
  $l' \leftarrow \mathrm{indexmax}_{1 \le l \le v}(T_l - H(L_l, E_l + \Delta E))$; (12)
  $E_{l'} \leftarrow E_{l'} + \Delta E$; (13)
  $T_{l'} \leftarrow H(L_{l'}, E_{l'})$; (14)
  $remainingE \leftarrow remainingE - \Delta E$; (15)
end do; (16)
$T \leftarrow T_1 + T_2 + \cdots + T_v$. (17)

The initial energy constraint E_l for L_l is determined as follows. Let us define $R_l = \sum_{t_i \in L_l} r_i$ and $D_l = \sum_{t_i \in L_l} d_i$, for all $1 \le l \le v$. Then, according to [13], we can set E_l as

$$E_l = R_l P_s^{1-1/\alpha} \xi^{1/\alpha} \frac{\alpha}{(\alpha - 1)^{1-1/\alpha}} + \left(\frac{\ln 2}{\min_{1 \le j \le n}(w_j \beta_j)} \right) D_l, \tag{31}$$

for all $1 \le l \le v$.

We would like to mention that for independent tasks in the same level, our heuristic energy-constrained scheduling algorithm H assigns the same computation speed s_0 to all tasks executed locally on the UE and the same communication speed c_j to all tasks executed remotely on MEC$_j$. However, tasks from different levels have different computation speeds even if they are all executed locally on the UE, and tasks from different levels have different communication speeds even if they are all executed remotely on the same MEC.

The time complexity of Algorithm 3 is analyzed as follows. From [13], we know that algorithm H takes $O(|L_l| n^2 \log(I/\epsilon))$ time in line (2). Thus, the for-loop in lines (1)–(3) takes $O(mn^2 \log(I/\epsilon))$ time, since $|L_1| + |L_2| + \cdots + |L_v| = m$. The most time consuming step in the while-loop of lines (6)–(16) is line (12), which takes $O(mn^2 \log(I/\epsilon))$ time. Therefore, the overall time complexity of Algorithm 3 is $O(Kmn^2 \log(I/\epsilon))$.

4.2 Time-Constrained Scheduling

In this section, we consider time-constrained scheduling with post-power-allocation.

Our time-constrained scheduling algorithm with post-power-allocation, called Time-Constrained Level-by-Level Scheduling with Heuristic H (TCLL-H), is presented in Algorithm 4.

Algorithm 4. Time-Constrained Level-by-Level Scheduling With Heuristic H (TCLL-H)

Input: $A = (L, \prec)$ with $L = (t_1, t_2, \ldots, t_m)$, where $t_i = (r_i, d_i)$, for all $1 \le i \le m$, UE $= (\xi, \alpha, P_s)$, MEC$_j = (s_j, w_j, \beta_j)$, for all $1 \le j \le n$, and $\tilde{T}$.
Output: A computation offloading strategy and a power allocation strategy such that T does not exceed $\tilde{T}$ and E is minimized.

for $(l = 1; l \le v; l{+}{+})$ **do** (1)
  $E_l \leftarrow H(L_l, T_l)$; (2)
end do; (3)
$additionalT \leftarrow (T_1 + T_2 + \cdots + T_v) - \tilde{T}$; (4)
$T' \leftarrow ((T_1 + T_2 + \cdots + T_v) - \tilde{T})/K$; (5)
while $(additionalT\ >\ 0)$ **do** (6)
  if $(additionalT \le T')$ **then** (7)
    $\Delta T \leftarrow additionalT$; (8)
  else (9)
    $\Delta T \leftarrow \gamma T'$, where $\gamma \in [0.5, 1.0]$; (10)
  end if; (11)
  $l' \leftarrow \mathrm{indexmin}_{1 \le l \le v}(H(L_l, T_l - \Delta T) - E_l)$; (12)
  $T_{l'} \leftarrow T_{l'} - \Delta T$; (13)
  $E_{l'} \leftarrow H(L_{l'}, T_{l'})$; (14)
  $additionalT \leftarrow additionalT - \Delta T$; (15)
end do; (16)
$E \leftarrow E_1 + E_2 + \cdots + E_v$. (17)

The key issue in level-by-level time-constrained scheduling is to determine how the given time budget $\tilde{T}$ is allocated to the v levels. Let $H(L_l, T_l)$ be the total energy consumption when algorithm H is applied to L_l with time constraint T_l. Initially, each level L_l is scheduled by using algorithm H with some initial time allocation T_l (lines (1)–(3)). Then, the additional time $(T_1 + T_2 + \cdots + T_v) - \tilde{T}$ (line (4)) is divided by K to get T' (line (5)), and the while-loop in lines (6)–(16) is repeated slightly more than K times. In each repetition, the following actions are performed. First, ΔT is determined, which is a random number γ times T', where γ is uniformly distributed in [0.5,1.0] (line (10)). Second, the level l' which results in the minimum increment in its total energy consumption if ΔT amount of time is reduced, i.e., $H(L_l, T_l - \Delta T) - H(L_l, T_l)$, is selected (line (12)). Third, the execution time of level l' is reduced by ΔT (line (13)). The while-loop terminates after all the additional time is reduced (line (6)). The total energy consumption E of the mobile application is simply $H(L_1, T_1) + H(L_2, T_2) + \cdots + H(L_v, T_v)$ (line (17)).

The initial time constraint T_l for L_l is determined as follows. According to [13], we can set T_l as

$$T_l = \frac{R_l}{\min(s_1, s_2, \ldots, s_n)}, \tag{32}$$

for all $1 \le l \le v$.

The time complexity of Algorithm 4 is analyzed as follows. From [13], we know that algorithm H takes $O(|L_l|n)$ time in line (2). Thus, the for-loop in lines (1)–(3) takes $O(mn)$ time, since $|L_1| + |L_2| + \cdots + |L_v| = m$. The most time consuming step in the while-loop of lines (6)–(16) is line (12), which takes $O(mn)$ time. Therefore, the overall time complexity of Algorithm 4 is $O(Kmn)$.

5 Experimental Performance Evaluation

We experimentally evaluate the performance of our proposed algorithms in this section.

5.1 Experiment Settings

A fog computing environment with one UE and $n = 7$ MECs is considered. The UE is configured with the following parameters: $\xi = 0.1$, $\alpha = 2.0$, $P_s = 0.05$ Watts. The MEC$_j$ is configured with the following parameters: $s_j = 3.1 - 0.1j$ BI/second, $w_j = 2.9 + 0.1j$ MB/second, $\beta_j = 2.1 - 0.1j$ Watts^{-1}, for all $1 \le j \le n$.

Task computation and communication requirements are randomly generated. The r_i's are independent and identically and uniformly distributed in the range [1.5,5.0]. The d_i's are independent and identically and uniformly distributed in the range [1.0, 3.0].

A *random directed acyclic graph* with m nodes and arc probability p is generated using the following procedure. For each pair of tasks t_{i_1} and t_{i_2}, where $1 \le i_1 < i_2 \le m$, an arc (t_{i_1}, t_{i_2}) exists with probability p. The arc probabilities are independent of each other. It is easy to see that the expected number of successors of task t_i is $(m - i)p$, where $1 \le i \le m$. If $p = b/m$, then it is in the range $[0, b)$. We set $b = 2$ in this section.

To show numerical characteristics of the above random dags, for $m = 20, 40, 60, \ldots, 200$, the expected number of levels $\overline{v}$, and the expected number of tasks (width) $\overline{m}_l$ on level l for $l = 1, 2, 3, 4$, are displayed below. These data are the averages of those collected from 5000 random dags. For all the data in the table, the maximum 99% confidence interval (C.I.) is ±2.59677%. It is observed that a random dag exhibits the shape of an inverted cone, i.e., the levels 1, 2, 3, 4,... have decreasing widths.

m	$\overline{v}$	$\overline{m}_1$	$\overline{m}_2$	$\overline{m}_3$	$\overline{m}_4$
20	4.426	8.756	5.389	3.309	1.684
40	5.151	17.450	10.460	6.469	3.444
60	5.541	26.128	15.594	9.593	5.197
80	5.823	34.731	20.758	12.674	6.968
100	6.054	43.434	25.691	15.830	8.765
120	6.192	52.061	30.860	19.020	10.527
140	6.341	60.654	36.013	22.160	12.340
160	6.463	69.414	41.013	25.316	14.049
180	6.583	77.900	46.110	28.415	15.825
200	6.644	86.725	51.206	31.549	17.610

The following heuristics for the initial order of $L = (t_{i_1}, t_{i_2}, \ldots, t_{i_m})$ are considered in this paper.

- ORG (*Original Order*) – Tasks are arranged in their original order.
- SRF (*Smallest Requirement First*) – Tasks are ordered in such a way that $r_{i_1} \le r_{i_2} \le \cdots \le r_{i_m}$.
- LRF (*Largest Requirement First*) – Tasks are ordered in such a way that $r_{i_1} \ge r_{i_2} \ge \cdots \ge r_{i_m}$.
- SDF (*Smallest Data First*) – Tasks are ordered in such a way that $d_{i_1} \le d_{i_2} \le \cdots \le d_{i_m}$.

TABLE 1
Experimental Data for Energy-Constrained List Scheduling (99% C.i. = ±2.73657%)

m	ORG	SRF	LRF	SDF	LDF	SRD	LRD	RAN10	RAN30	RAN50
20	6.33308	6.48709	6.66739	6.55910	6.42811	6.40151	6.59637	5.90977	5.78460	5.74500
40	9.11599	10.04377	9.93155	10.01859	9.86338	9.95566	9.95979	8.56220	8.34164	8.26095
60	12.33894	13.63935	13.55841	13.58227	13.44811	13.48620	13.57923	11.84143	11.62201	11.53058
80	15.73861	17.09231	17.19463	17.24645	17.09973	17.10994	17.11351	15.29914	15.11473	15.03161
100	19.36595	20.75231	20.72037	20.97696	20.84537	20.82771	20.80411	18.90222	18.68869	18.60861
120	23.02285	24.54459	24.47294	24.65159	24.41779	24.42716	24.46665	22.58554	22.37142	22.27217
140	26.55621	28.01454	28.00879	28.23518	28.05862	28.01719	28.05040	26.16482	25.95939	25.86265
160	30.22643	31.62066	31.70171	31.84462	31.65625	31.38314	31.80101	29.79547	29.56064	29.46299
180	33.87471	35.42949	35.49348	35.64393	35.57105	35.43361	35.49940	33.46656	33.24222	33.14254
200	37.49787	38.87911	38.89904	39.16259	39.01920	38.92612	39.03835	37.07450	36.84821	36.76140

TABLE 2
Experimental Data for Time-Constrained List Scheduling (99% C.i. = ±3.27861%)

m	ORG	SRF	LRF	SDF	LDF	SRD	LRD	RAN10	RAN30	RAN50
20	9.92750	9.96305	10.21354	10.30031	9.72279	9.71818	10.42555	7.68352	7.50296	7.43569
40	13.46612	14.37904	14.37946	14.60144	14.33305	14.32447	14.51679	12.72746	12.52871	12.44246
60	18.98870	20.31615	20.14062	20.24720	20.11653	20.26532	20.24780	18.57314	18.38512	18.30668
80	24.97693	26.24193	26.14677	26.20386	26.10228	26.32500	26.30230	24.62434	24.43361	24.34567
100	31.17557	32.41219	32.41821	32.23753	32.33055	32.65344	32.55194	30.79719	30.59395	30.49837
120	37.39229	38.88093	38.70792	38.46860	38.73005	39.04237	38.86885	37.00367	36.77652	36.67288
140	43.66139	45.19537	45.03864	44.68140	44.94225	45.41580	45.23217	43.24306	42.99051	42.88581
160	50.04367	51.70387	51.49461	51.03058	51.37655	51.94286	51.85121	49.60160	49.33935	49.22079
180	56.52746	58.22899	58.00216	57.36592	57.94094	58.57126	58.36803	55.95753	55.69383	55.57383
200	62.97426	64.87100	64.53795	63.74898	64.30659	65.15472	64.94806	62.32742	62.03703	61.89727

- LDF (*Largest Data First*) – Tasks are ordered in such a way that $d_{i_1} \geq d_{i_2} \geq \cdots \geq d_{i_m}$.
- SRD (*Smallest Requirement-Data-Ratio First*) – Tasks are ordered in such a way that $r_{i_1}/d_{i_1} \leq r_{i_2}/d_{i_2} \leq \cdots \leq r_{i_m}/d_{i_m}$.
- LRD (*Largest Requirement-Data-Ratio First*) – Tasks are ordered in such a way that $r_{i_1}/d_{i_1} \geq r_{i_2}/d_{i_2} \geq \cdots \geq r_{i_m}/d_{i_m}$.
- RANk (*Best of k Random Orders*) – Tasks are arranged in k random orders and the best of the k solutions are taken. We set $k = 10, 30, 50$.

5.2 Evaluation of Pre-Power-Allocation Algorithms

In this section, we examine the performance of pre-power-allocation algorithms.

In Table 1, we display our experimental results for energy-constrained list scheduling. We set $m = 20, 40, 60, \ldots, 200$ for the number of tasks, and $\tilde{E} = 4 + 8(m/10)$ Joules for the energy constraint. For each m, we generate $M = 500$ random directed acyclic graphs with m nodes and arc probability $p = 2/m$. For each random dag, we employ the ten proposed heuristic algorithms, i.e., ECLS-H with $H =$ ORG, LRF, SRF, LDF, SDF, LRD, SRD, RAN10, RAN30, and RAN50. The average of the M results of each heuristic algorithm is shown in the table. For all the data in the table, the maximum 99% C.I. is ±2.73657%.

In Table 2, we display our experimental results for time-constrained list scheduling. We set $m = 20, 40, 60, \ldots, 200$ for the number of tasks, and $\tilde{T} = 3 + 3(m/10)$ seconds for the time constraint. (We set $\tilde{E} = 6 + 4(m/10)$ in line (1) and $\Delta E = 1$ in line (4).) For each m, we generate $M = 1000$ random directed acyclic graphs with m nodes and arc probability $p = 2/m$. For each random dag, we employ the ten proposed heuristic algorithms, i.e., TCLS-H with $H =$ ORG, LRF, SRF, LDF, SDF, LRD, SRD, RAN10, RAN30, and RAN50. The average of the M results of each heuristic algorithm is shown in the table. For all the data in the table, the maximum 99% C.I. is ±3.27861%.

From Tables 1 and 2, we can make the following important observations.

- The heuristics LRF, SRF, LDF, SDF, LRD, SRD do not yield noticeable difference in performance. Surprisingly, even ORG performs better than LRF, SRF, LDF, SDF, LRD, SRD.
- The strategy of repeating the algorithm multiple times does yield performance improvement. RAN10 performs noticeably better than ORG, LRF, SRF, LDF, SDF, LRD, SRD. However, excessive repetition does not bring much benefit, e.g., RAN30 and RAN50 do not perform noticeably better than RAN10.

5.3 Evaluation of Post-Power-Allocation Algorithms

In this section, we examine the performance of post-power-allocation algorithms.

In Table 3, we display our experimental results for energy-constrained level-by-level scheduling. We set $m = 20, 40, 60, \ldots, 200$ for the number of tasks, and $\tilde{E} = 4 + 8(m/10)$ Joules for the energy constraint. For each m, we generate $M = 200$ random directed acyclic graphs with m nodes and arc probability $p = 2/m$. For each random dag, we employ the ten proposed heuristic algorithms, i.e., ECLL-H with $H =$ ORG, LRF,

TABLE 3
Experimental Data for Energy-Constrained Level-by-Level Scheduling (99% C.i. = ±3.66418%)

m	ORG	SRF	LRF	SDF	LDF	SRD	LRD	RAN10	RAN30	RAN50
20	6.14116	6.41678	5.63241	6.31427	6.01120	6.25182	5.97162	5.58039	5.54089	5.52361
40	9.39722	9.62249	8.77565	9.68777	9.17091	9.37413	9.11474	8.62037	8.54037	8.51479
60	12.62807	12.75280	11.91830	13.01803	12.24451	12.39116	12.33286	11.70112	11.61171	11.57667
80	15.81300	15.77216	15.16399	16.27605	15.31049	15.37886	15.58691	14.87892	14.76326	14.72709
100	19.07778	18.94351	18.46658	19.73619	18.55745	18.52221	18.99568	18.14137	18.04773	18.01506
120	22.32952	21.99955	21.81596	23.06160	21.78133	21.59613	22.36492	21.38840	21.29001	21.24946
140	25.46007	24.99503	25.00448	26.31298	24.83862	24.55783	25.60256	24.51529	24.42089	24.37641
160	28.76626	28.11156	28.29632	29.68970	28.04417	27.66150	28.99075	27.77727	27.66976	27.63487
180	31.97630	31.22202	31.58380	33.04419	31.23029	30.72951	32.34895	31.00756	30.91417	30.88518
200	35.94577	34.98050	35.55188	37.11161	35.12320	34.55036	36.43820	34.95769	34.86119	34.81617

TABLE 4
Experimental Data for Time-Constrained Level-by-Level Scheduling (99% C.i. = ±4.32958%)

m	ORG	SRF	LRF	SDF	LDF	SRD	LRD	RAN10	RAN30	RAN50
20	7.17612	7.45978	6.95541	7.31397	7.07785	7.27085	7.15486	6.91344	6.87709	6.86554
40	12.33779	12.67933	12.24361	12.52658	12.24655	12.49158	12.55159	12.05454	12.02382	12.01221
60	17.90778	18.28847	17.94457	18.11941	17.81361	18.17692	18.40052	17.63386	17.60367	17.59392
80	23.64200	24.08126	23.77834	23.84809	23.59230	24.06870	24.35092	23.36042	23.33168	23.32336
100	29.17348	29.68381	29.43206	29.39553	29.17235	29.71204	30.16238	28.90701	28.87904	28.86922
120	34.85638	35.43268	35.18821	35.11733	34.86430	35.54556	36.11573	34.58585	34.55644	34.54599
140	40.41645	41.06815	40.87987	40.66464	40.44734	41.26272	41.94877	40.13437	40.10787	40.09642
160	46.13776	46.82764	46.67916	46.42114	46.18082	47.10550	47.96953	45.84501	45.81365	45.80240
180	51.76933	52.57947	52.48760	52.09895	51.86871	52.94790	53.92911	51.50235	51.47520	51.46351
200	59.16643	60.60973	59.91616	59.54336	59.42100	61.31686	61.79147	58.75088	58.68758	58.66142

SRF, LDF, SDF, LRD, SRD, RAN10, RAN30, and RAN50. The parameter K is set as 10. The average of the M results of each heuristic algorithm is shown in the table. For all the data in the table, the maximum 99% C.I. is ±3.66418%.

In Table 4, we display our experimental results for time-constrained level-by-level scheduling. We set $m = 20, 40, 60, \ldots, 200$ for the number of tasks, and $\tilde{T} = 3 + 3(m/10)$ seconds the time constraint. For each m, we generate $M = 400$ random directed acyclic graphs with m nodes and arc probability $p = 2/m$. For each random dag, we employ the ten proposed heuristic algorithms, i.e., TCLL-H with $H =$ ORG, LRF, SRF, LDF, SDF, LRD, SRD, RAN10, RAN30, and RAN50. The parameter K is set as 200. The average of the M results of each heuristic algorithm is shown in the table. For all the data in the table, the maximum 99% C.I. is ±4.32958%.

From Tables 3 and 4, we can make the following important observations.

- Different heuristics do yield noticeable difference in performance. First, when m is small (large, respectively), i.e., when $m \le 100$ ($m > 100$, respectively), LRF (SRD, respectively) is the best heuristic among ORG, LRF, SRF, LDF, SDF, LRD, SRD for energy-constrained level-by-level scheduling. Second, LDF is the best heuristic among ORG, LRF, SRF, LDF, SDF, LRD, SRD for time-constrained level-by-level scheduling.
- The strategy of repeating the algorithm multiple times does not yield much performance improvement. For instance, the performance of SRD and LDF are already very close to that of RAN50 for energy-constrained and time-constrained level-by-level scheduling respectively.

5.4 Comparison

It is observed that post-power-allocation algorithms consistently outperform pre-power-allocation algorithms in almost all cases. Although the list scheduling algorithm is very effective and efficient in handling precedence constraints, the equal-energy method for pre-power-allocation is not efficient. Although the level-by-level scheduling method is not as efficient as the list scheduling algorithm, the computation offloading strategies and the post-power-allocation strategies developed in [13] for independent tasks in the same level are very effective and efficient.

6 Related Research

We review related research in this section.

In recent years, extensive investigation has been conducted for computation offloading in mobile edge computing and fog computing, which has been a very active and productive research area. Refs. [1], [10], [20] provide recent comprehensive surveys.

Scheduling precedence constrained tasks for mobile applications in mobile edge computing and fog computing has been investigated by several researchers (see Table 5). Almost in all existing studies, only the case of one UE and one MEC has been considered. Therefore, the *where* issue in a computation offloading strategy becomes a *whether* issue, i.e., a binary computation offloading decision (either local or remote execution), and task scheduling is conducted only

TABLE 5
Research in Computation Offloading for Mobile Applications With Precedence Constrained Tasks (✓: Considered; –: Not Considered)

Work	Precedence Constraint	Multiple MECs	Computation Speed	Communication Speed	Makespan	Energy Constraint
Ref. [2]	✓	–	–	–	✓	✓
Ref. [4]	✓	–	✓	✓	–	✓
Ref. [6]	✓	–	–	–	✓	–
Ref. [9]	✓	–	–	✓	✓	✓
Ref. [14]	✓	–	✓	–	✓	–
Ref. [19]	✓	–	–	✓	–	✓
Ref. [21]	✓	–	–	–	✓	✓
This paper	✓	✓	✓	✓	✓	✓

on a UE and an MEC. A power allocation strategy still involves the computation speed for local execution and/or the communication speed remote execution. The main performance measure is the overall execution time (i.e., the makespan, or the maximum completion time of all tasks) of a mobile application. Sometimes, the summation of execution times (or completion times) of all tasks is also used as a performance measure. The main cost measure has been unanimously the total energy consumption of a mobile application.

By manipulating only the computation offloading decision, Deng *et al.* minimized the total energy consumption while satisfying a strict delay (i.e., makespan) constraint using a particle swarm optimization algorithm [2]. Based on computation offloading decision, transmission power allocation, and clock frequency control, Guo *et al.* minimized the summation of each task's weighted sum of completion time and energy consumption [4]. Jia *et al.* presented an online task offloading algorithm on a mobile device to minimize the completion time of an application [6]. By making computation offloading decision and transmission power selection, Khalili and Simeone minimized a weighted sum of total energy consumption and overall latency (i.e., makespan) [9]. By computation offloading adjusting and frequency scaling, Liang *et al.* tried to minimize the makespan over an MEC center with multiple servers [14]. By making computation offloading decision and transmission power selection, Lorenzo *et al.* minimized the energy consumption at the mobile site, under a power budget constraint and a latency constraint, where "latency" is the summation of task transfer and execution times [19]. By manipulating only the computation offloading decision, Mahmoodi *et al.* maximized the energy saved through remote execution, with a runtime deadline constraint, i.e., the completion time of the last component (i.e., makespan) does not exceed a time deadline [21].

The above literature review reveals several major weaknesses of current research. First, only a single MEC is considered, which makes the computation offloading decision much simpler and eliminates the challenging MEC selection problem. Second, some researchers adopt the summation of task execution times, not the makespan, as the performance measure, which not only makes less sense, but also simplifies the problem. Third, computation offloading should be conducted together with power allocation for computation and communication speeds and energy constraint. For these reasons, there is lack of investigation of combinatorial optimization approach to computation offloading within a framework similar to that of traditional energy-efficient task scheduling. In this paper, we have considered multiple heterogeneous MECs which have different computation speeds and communication speeds. We have also employed the makespan as the performance measure, which is the main objective of optimization in traditional task scheduling and the main concern for a mobile application consisting of tasks connected by a directed acyclic graph. Furthermore, we have incorporated power allocation and energy constraint into consideration.

Some researchers have explored related but different situations and environments. A fully polynomial time approximation scheme was proposed by Kao *et al.* to find a task assignment strategy on multiple devices, so as to minimize the cost constrained latency [7]. Lin *et al.* considered a mobile device with multiple heterogeneous cores and minimized the total energy consumption under a task completion time (i.e., makespan) budget, i.e., a delay constraint, by making computation offloading decision and determining heterogeneous cores mapping, execution frequency of each local task, schedule of the tasks on heterogeneous cores and the MEC [15]. Liu *et al.* investigated task offloading with both precedence and placement constraints in a multi-user MEC environment based on spatio-temporal information of tasks and servers [16]. Liu *et al.* minimized the total weighted cost of energy and delay in a multiple MEC environment by incorporating the mobility of a mobile device into consideration [17]. Long *et al.* studied one MEC and one cloud server, i.e., there are three (local, edge, cloud) computation models for each task, and minimized the total energy consumption under an application completion time (i.e., makespan) constraint by manipulating only computation offloading decision [18]. Yang *et al.* concerned multiple UEs and one MEC with multiple homogeneous servers, where the computation offloading decision needs to determine where (including the mobile device and the cloud servers) to execute a task, and minimized the average application delay of the users, where the dags are linear and sequential dags and energy consumption is not considered [24].

We would like to mention that there are studies focusing on hierarchical fog computing environments [5], [8], [22]. These work mainly paid attention on the structure of a multi-level fog computing network, not the structure of a mobile application.

7 Concluding Remarks

In this paper, we have addressed scheduling precedence constrained tasks of a mobile application in a fog computing environment. We have developed the class of pre-power-allocation algorithms and the class of pre-power-allocation algorithms. We have also experimentally evaluated the proposed algorithms, and found that ECLL-LRF and ECLL-SRD are the best algorithms for energy-constrained scheduling, and TCLL-LDF is the best algorithm for time-constrained scheduling.

There are several research directions worth of further exploration. First, there is still room for performance improvement by considering more sophisticated and efficient computation offloading strategies and power allocation strategies and new algorithmic schemes different from pre-power-allocation algorithms and post-power-allocation algorithms. Second, it is definitely interesting and challenging to analyze the performance of heuristic algorithms when compared with optimal solutions. So far, little result is known in this area even for independent tasks [13], and much efforts and insights are required to bring breakthrough and significant advancement.

Acknowledgments

The author would like to express his gratitude to the anonymous reviewers for their criticism and comments on the manuscript.

References

[1] A. Bhattacharya and P. De, "A survey of adaptation techniques in computation offloading," *J. Netw. Comput. Appl.*, vol. 78, pp. 97–115, 2017.
[2] M. Deng, H. Tian, and B. Fan, "Fine-granularity based application offloading policy in cloud-enhanced small cell networks," *Proc. IEEE Int. Conf. Commun. Workshops*, 2016, pp. 638–643.
[3] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM J. Appl. Math.*, vol. 2, pp. 416–429, 1969.
[4] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
[5] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments," *Softw.: Pract. Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.
[6] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2014, pp. 352–357.
[7] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Trans. Mobile Comput.*, vol. 16, no. 11, pp. 3056–3069, Nov. 2017.
[8] A. Kaur and N. Auluck, "Scheduling algorithms for hierarchical fog networks," Dec. 09, 2021. [Online]. Available: https://arxiv.org/abs/2112.04715
[9] S. Khalili and O. Simeone, "Inter-layer per-mobile optimization of cloud mobile computing: A message-passing approach," *Trans. Emerg. Telecommun. Technol.*, vol. 27, no. 6, pp. 814–827, 2016.
[10] M. A. Khan, "A survey of computation offloading strategies for performance improvement of applications running on mobile devices," *J. Netw. Comput. Appl.*, vol. 56, pp. 28–40, 2015.
[11] K. Li, "Power allocation and task scheduling on multiprocessor computers with energy and time constraints," in *Energy-Efficient Distributed Computing Systems*, A. Y. Zomaya and Y. C. Lee, Eds., Hoboken, NJ,USA: Wiley, 2012, pp. 1–37.
[12] K. Li, "Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers," *IEEE Trans. Comput.*, vol. 61, no. 12, pp. 1668–1681, Dec. 2012.
[13] K. Li, "Heuristic computation offloading algorithms for mobile users in fog computing," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 2, 2021, Art no. 11.
[14] J. Liang, K. Li, C. Liu, and K. Li, "Joint offloading and scheduling decisions for DAG applications in mobile edge computing," *Neurocomputing*, vol. 424, pp. 160–171, 2021.
[15] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment," *IEEE Trans. Serv. Comput.*, vol. 8, no. 2, pp. 175–186, Mar./Apr. 2015.
[16] B. Liu, X. Xu, L. Qi, Q. Ni, and W. Dou, "Task scheduling with precedence and placement constraints for resource utilization improvement in multi-user MEC environment," *J. Syst. Archit.*, vol. 114, 2021, Art. no. 101970.
[17] Y. Liu, C. Liu, J. Liu, Y. Hu, K. Li, and K. Li, "Mobility-aware and code-oriented partitioning computation offloading in mobile edge computing," *J. Grid Comput.*, vol. 20, 2022, Art. no. 11.
[18] X. Long, J. Wu, and L. Chen, "Energy-efficient offloading in mobile edge computing with edge-cloud collaboration," in *Proc. Int. Conf. Algorithms Architect. Parallel Process.*, 2018, pp. 460–475.
[19] P. D. Lorenzo, S. Barbarossa, and S. Sardellitti, Joint optimization of radio resources and code partitioning in mobile edge computing, Feb. 03, 2016. [Online]. Available: https://arxiv.org/abs/1307.3835
[20] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surv. Tut.*, vol. 19, no. 3, pp. 1628–1656, July.–Sep. 2017.
[21] S. E. Mahmoodi, R. N. Uma, and K. P. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 301–313, Apr.–Jun. 2019.
[22] M. Peixoto, T. Genez, and L. F. Bittencourt, "Hierarchical scheduling mechanisms in multi-level fog computing," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2021.3079110.
[23] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling interactive perception applications on mobile devices," in *Proc. 9th Int. Conf. Mobile Syst., Appl., Serv.*, 2011, pp. 43–56.
[24] L. Yang, J. Cao, H. Cheng, and Y. Ji, "Multi-user computation partitioning for latency sensitive mobile cloud applications," *IEEE Trans. Comput.*, vol. 64, no. 8, pp. 2253–2266, Aug. 2015.

Keqin Li (Fellow, IEEE) is a SUNY distinguished professor of computer science with the State University of New York. He is also a national distinguished professor with Hunan University, China. His current research interests include cloud computing, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber-physical systems, heterogeneous computing systems, Big Data computing, high-performance computing, CPU-GPU hybrid and cooperative computing, computer architectures and systems, computer networking, machine learning, intelligent and soft computing. He has authored or coauthored more than 850 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He holds more than 70 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world's top 5 most influential scientists in parallel and distributed computing in terms of both single-year impact and career-long impact based on a composite indicator of Scopus citation database. He has chaired many international conferences. He is currently an associate editor of the *ACM Computing Surveys* and the *CCF Transactions on High Performance Computing*. He has served on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *IEEE Transactions on Cloud Computing*, the *IEEE Transactions on Services Computing*, and the *IEEE Transactions on Sustainable Computing*. He is a fellow of the Asia-Pacific Artificial Intelligence Association (AAIA). He is also a member of Academia Europaea (Academician of the Europe).