IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 19, 2019, accepted December 10, 2019, date of publication December 23, 2019,
date of current version January 2, 2020.

Digital Object Identifier 10.1109/ACCESS.2019.2961453

A Fast Algorithm for Energy-Saving Offloading With Reliability and Latency Requirements in Multi-Access Edge Computing

HAOLIN LIU[1,2,3], LE CAO[1], TINGRUI PEI[1,2], QINGYONG DENG[1,2], (Member, IEEE), AND JIANG ZHU[1,2]

[1]College of Information Engineering, Xiangtan University, Xiangtan 411105, China
[2]Key Laboratory of Hunan Province for Internet of Things and Information Security, Xiangtan University, Xiangtan 411105, China
[3]Postdoctoral Research Station for Mechanics, Xiangtan University, Xiangtan 411105, China

Corresponding author: Tingrui Pei (peitingrui@xtu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61902336 and Grant 61672447, and in part by the Natural Science Foundation of Hunan Province, China, under Grant 2017JJ3316 and Grant 2019JJ50592.

ABSTRACT Multi-Access Edge Computing (MEC) is a promising paradigm that providing cloud-like service for handling the high-complexity and latency-sensitive applications on user equipment (UE) via computation offloading. However, the execution reliability is rarely considered in current MEC studies, which is an important factor to guarantee the quality of service (QoS). For that, this paper considers an energy-saving offloading to satisfy the reliability and latency requirements of the application. Specifically, we formulate an optimization problem to minimize the UE's energy consumption with reliability and latency constraints. To tackle this NP-hard problem, we first divide the entire application into multiple directed-acyclic-graph-(DAG)-based subtasks, where the subtask can be executed on the UE locally or MEC server remotely. Then, we decompose the overall reliability and latency requirements into multiple constraints for each subtask. Finally, we propose a fast heuristic algorithm to find a solution satisfying the constraints. Simulation results demonstrate the proposed algorithm obtains lower energy consumption compared with the local execution and random assignment and costs less runtime compared with the greedy algorithm.

INDEX TERMS Multi-access edge computing, computation offloading, energy consumption minimization, reliability guarantee.

I. INTRODUCTION

In recent years, with the development and popularity of mobile devices and mobile Internet traffic [1], [2], more and more applications, such as augmented reality, virtual reality, face recognition, mobile healthcare, and interactive games, gradually go deep into our lives [3], [4]. However, due to the limited battery capacity and computation resources [5], it is difficult for the mobile devices to run the high-complexity applications with a satisfactory quality of experience (QoE) and quality of service (QoS). Although Cloud Computing provides a solution to this problem, it leads to the heavy load and congestion in the network which influences the latency-sensitive applications [4]. It is estimated that tens of billions of edge devices with growing computation capability will be deployed in the near future. A novel paradigm named as Multi-Access Edge Computing (MEC) can be envisioned as a promising technique for taking advantage of the computation and storage resources at the edge of the network to provide cloud-like service for handling the high-complexity and latency-sensitive applications on user equipment (UE) via computation offloading [3].

To guarantee the QoS, there have been several studies on MEC. Especially, latency, can be regarded as the response time of an application or a task, is always considered [3], [4]. Reliability, also known as robustness in some aspects, plays a critical role in the QoS guarantee for applications in cloud computing [7]. However, the reliability requirement of the application in the field of MEC has received little attention. In the cloud-based system, reliability is defined as the probability that the scheduling is successfully completed for an application [7], which can also be applied to the

The associate editor coordinating the review of this manuscript and approving it for publication was Zhongming Zheng.

field of MEC [8]. In practice, due to the presence of faults in the hardware and software system, the 100% reliability cannot be easily achieved in any application [8]. So, if an application achieves its certified reliability goals, it can be considered reliable in a security-critical scenario. Reliability goals have been set in many industrial safety standards, such as the ISO 26262 standard for automotive software systems, the DO-178C standard for avionics software, and the IEC 61508 standard for various industrial software systems [9], [10]. In the combination of MEC and some security-critical applications (e.g., autonomous driving, Internet of vehicles, smart home, etc.), if the reliability goals of these applications are not achieved, it may lead to the serious consequences. Therefore, reliability is an important safety factor that should be guaranteed for security-critical applications in MEC.

Due to the faults in execution and communication, repeating execution or repeating transmission many times is an easy way to meet the reliability requirements. However, this approach not only takes a lot of time but also causes energy waste, which is not a good way for the UE whose energy resources are limited [11], [12]. For that, according to references [13], [14], we mainly consider the reliability during execution, for which the task of an application is divided into multiple sub-tasks. It would reduce the complexity of running the entire application and decrease execution latency by the parallel processing, and then the sub-tasks will be offloaded to the various MEC servers or performed locally to guarantee the task's overall reliability.

Routinely, the UE's resource is often limited in MEC and should be optimized via the computation offloading. In the computation offloading, the tasks in UE can be executed locally or on the MEC server. The task's execution on the MEC server can provide low execution latency, high reliability, unlimited energy supply that is no need to worry. But even so, the latency and energy cost during the transmission cannot be ignored. Therefore, a tradeoff between the latency, the reliability, and the energy consumption should be addressed. For that, we consider a scenario to save the UE's energy by optimizing the offloading decision in which the latency and reliability requirements are satisfied.

The main contributions of our work are summarized as follows:

1) To guarantee the satisfactory reliability of the entire task of an application, it is divided into multiple subtasks that can be offloaded to the MEC servers for execution. It avoids the repeating execution of the task in a single UE for the reliability and would save the UE's energy. For an independent task, its subtasks may depend on other subtasks after it is divided. Thus, we use the directed acyclic graph (DAG) to represent the subtasks' relationship [14], which not only accurately represents the dependencies of each subtask, but also figures out the execution order of these subtasks.

2) We formulate an integer programming problem with the constraints of reliability and latency for the entire application. Since it is difficult to find a solution at once to satisfy the overall constraints, we decompose the problem into three sub-problems, and the overall reliability and latency requirements are also divided into multiple constraints for each subtask.

3) We propose a fast heuristic algorithm by adopting a greedy strategy to make the offloading decision of each subtask, which can achieve lower energy consumption and meet its constraints. Compared with other algorithms, the proposed algorithm can reduce more unnecessary energy consumption with lower time complexity.

The rest of the paper is organized as follows. Section II gives brief overviews of some related works. In Section III, we introduce the system models. Section IV defines the problem. Section V proposes the heuristic algorithm for the optimization problem. In Section VI, we conduct some simulations to compare the proposed algorithm with a greedy algorithm, the local execution, and random assignment. Finally, we conclude this paper in Section VII.

II. RELATED WORKS

In recent years, various research works on computation offloading in the MEC have been proposed. For example, Rodrigues *et al.* [15] proposed a computation offloading scheme to determine whether the latency-critical tasks should be executed locally on the UE or remotely on the MEC servers. Wang *et al.* [16] studied the dynamic service migration problem and used the Markov decision process to solve this problem. Chen *et al.* [17] considered the energy and the latency as the overhead and adopted a distributed game-theoretic approach to reduce the overall network overhead. Mu *et al.* [18] jointly considered the application partitioning and collaborative computation offloading, so that UEs may help each other on task execution to meet the completion deadline of the applications while minimizing the overall energy consumption. Meskar *et al.* [19] constructed a network where users share the communication channel to offload their computations as a competitive game, and obtained an optimum offloading decision for minimizing user energy consumption while satisfying the deadline of the applications. Ren *et al.* [20] investigated the joint communication and resource allocation problem in a multiuser MEC system to minimize latency. Chen and Hao [21] studied the ultra-dense network's task offloading problem, whose goal is to minimize the delay while saving the energy cost of the UEs.

The above studies mainly investigated how to reduce energy consumption and latency during the computation offloading in MEC, while the reliability requirement of the application, which is also an important factor to guarantee the QoS, is necessary to be considered. Recent studies about the reliability in MEC are shown below: Liu *et al.* [22] proposed a novel partially-offloading scheme design to account for the ultra-reliable low-latency requirements of mission-critical applications in MEC. The ultra-reliable low-latency communication is one of the pillars in 5G. They considered the statistics of the extreme queue length as a reliability and latency measure and proposed a Lyapunov optimization method to solve the network-wide power minimization problem.

Liu and Zhang [23] considered the tradeoff between the latency and reliability in task offloading to MEC. They formulated an optimization problem to jointly minimize the latency and offloading failure probability and designed three algorithms on heuristic search, reformulation linearization technique and semi-definite relaxation to solve the problem. Han *et al.* [24] also considered the ultra-reliable and ultra-low-latency MEC services in the 5G telecommunication network. They mainly focused on the mobile network security for authentication and proposed a novel decentralized authentication architecture that supports flexible and low-cost local authentication with the awareness of context information of network elements. Different from [22]–[24] which are based on the ultra-reliable and ultra-low-latency service in 5G and focus on the communication reliability, our work pays more attention to the execution reliability of the edge devices and the internal relationship of subtasks. The execution reliability is rarely considered in MEC but is always concerned in the field of embedded systems. For example, Alsenani *et al.* [8] proposed a probabilistic system named as SaRa to estimate the reliability of untrusted edge resources in the volunteer cloud. Xie *et al.* [25] proposed three hardware cost optimization algorithms for functional safety-critical parallel applications on heterogeneous distributed embedded systems during the design phase. Xie *et al.* [13] designed enough replication for the redundancy minimization (ERRM) algorithm to satisfy the application's reliability requirement with low time complexity.

Due to the latency and energy requirements, these preceding algorithms for the execution reliability cannot directly be used in the MEC scenario. But the scheduling of the DAG-based tasks can be adopted to represent the distribution process in the offloading scheme, and additionally, the overall execution reliability of the application can be achieved by guaranteeing the execution reliability of each distributed subtask. Based on that, we formulate an optimization problem for minimizing the energy consumption of the UE with the latency and reliability constraints by the sub-tasks' scheduling.

III. SYSTEM MODELS

In this section, the system models will be described. We first introduce the edge network model and task model, and then specify the communication model and computation model in detail.

A. NETWORK MODEL

To enable tractable analysis, like the studies on partial offloading scheduling (e.g., [17], [26]), we consider a single-user MEC system and concentrate on the offloading scheduling of the DAG-based subtasks in a single UE.

In addition, we assume that there are some macro base stations deployed in the network, and the MEC server is directly embedded on the macro base station. So that the communication time between the macro base station and its own MEC server can be ignored. A series of processors on MEC servers are described as $\mathbb{S} = \{s_1, s_2, \ldots, s_m\}$. The UE is also a computing device for task execution in MEC, whose processor is denoted as s_0.

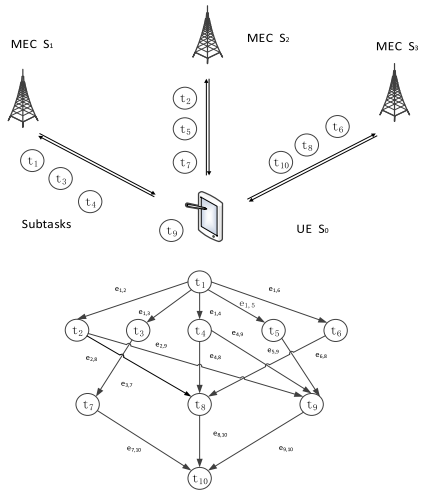


FIGURE 1. The illustration of the DAG-based task model.

As the same as cloud computing, we consider that the overall network will keep stable during the transmission and computation process in a slot (e.g. within a few seconds) [15]. In each slot, the UE has a computationally intensive and latency sensitive application to be performed. Its reliability and latency requirements are R_{req} and L_{req}, respectively. The UE can perform the application on the MEC server, or perform it on itself, depending on the offloading decision to satisfy the requirement of reliability and latency. Inspired by [14] and [25], we split the entire task of the application into some interdependence subtasks, and then change the single task assignment problem into a multi-task offloading problem to improve the system's concurrency.

B. TASK MODEL

In each slot, the task of the application is split into n subtasks whose relationship can be represented as a DAG $G = \{\mathbb{T}, \mathbb{E}\}$. The subtasks in G can be denoted as a set of $\mathbb{T} = \{t_1, t_2, \ldots, t_n\}$, and each subtask has its computation parameter $\{\omega_i, \mu_i\}$, ω_i (in Megacycles) denotes the total number of processor cycles required to finish the execution of task t_i, and μ_i (in KB) stands for the size of input data of task t_i, contains program codes and input parameters. $\mathbb{E} = \left\{e_{ik}\left(s_j, s_l\right)\right\}$ represents a series of connected edges in G. As shown in Figure 1, each edge represents the connection between two subtasks, and the direction indicates a constraint that the successor subtasks should not start until the predecessor subtasks completely be finished [14]. For the sake of convenience, we use $pred(t_i)$ and $succ(t_i)$ to represent a set of the immediate predecessor subtasks and the immediate successor subtasks of t_i, respectively. $e_{ik}\left(s_j, s_l\right)$ represents the communication cost between subtask t_i and subtask t_k. Since the communication cost is associated with the processors on the MEC server and the UE itself, $e_{ik}\left(s_j, s_l\right)$ is a function related to t_i's processor s_j and t_k's processor s_l. If t_i and t_k are not connected, then for arbitrary s_j and s_l, $e_{ik}\left(s_j, s_l\right) = 0$.The subtask without

the predecessor subtask is called the entry subtask t_{entry}, and the subtask without the successor subtask is called the exit subtask t_{exit}.

C. COMMUNICATION MODEL

Each macro base station in the network is linked to an MEC server, so the UE could offload its subtasks to the MEC server via the connected macro base station. In addition, the offloading data rate for a subtask t_i offloaded to the MEC server s_j is assumed as v_j, which can be expressed as

$$v_j = W \log_2(1 + \frac{Pg_j}{\sigma^2}) \tag{1}$$

where W denotes the channel bandwidth. P indicates the transmission power of UE. σ^2 denotes the noise power. g_j denotes the channel gain between the UE and server s_j,which can be defined as

$$g_j = d_j^{-\alpha} \tag{2}$$

where d_j is the distance between the UE and MEC server s_j, α is the path loss factor.

Then, we can get the transmission latency of subtask t_i offloaded to the server s_j:

$$L_{ij}^r = \mu_i / v_j \tag{3}$$

The energy consumed of the UE to offload subtask t_i is

$$\varepsilon_{ij}^r = PL_{ij}^r \tag{4}$$

D. COMPUTATION MODEL

A subtask can be executed on the UE itself, or be offloaded to the MEC server for execution. The process of executing a subtask locally or remotely on the MEC server is discussed below.

1) Local computing. For the local computing subtask, we define F_0 as the UE's processor computational capability whose most common unit of measure is processor cycles per second. Therefore, the local computation latency for the subtask t_i can be expressed as

$$L_i^l = \omega_i / F_0 \tag{5}$$

The energy consumed by t_i's local computing is given by

$$\varepsilon_i^l = \rho L_i^l \tag{6}$$

ρ indicates the energy consumed by the processor per unit time.

2) Offloading computing. Due to the supply from the power grid, the MEC server's energy cost is ignored in our work. Thus, in this part, we only consider the latency during the offloading process. $\mathbb{F} = \{F_1, F_2, \ldots, F_m\}$ is defined as the computational capability (processor cycles per second) of the MEC server. The latency for finishing an offloaded subtask contains the transmission latency and computation latency on the server. Due to the high bandwidth of the downlink channel and small size of the output data, the result's return time of the offloaded subtask is ignored. So the total latency of the subtask t_i executed on the server is

$$L_{ij}^o = \omega_i / F_j + L_{ij}^r \tag{7}$$

For the sake of simplicity, the latency of the subtask executed on the UE or MEC server is uniformly represented by L_{ij}

$$L_{ij} = \omega_i / F_j + L_{ij}^r \tag{8}$$

where $L_{ij}^r = 0$ when $j = 0$.

TABLE 1. Reliability requirement for different exposures in IEC 61508.

Exposure	Reliability Requirement
E1Very low probability (Not specified)	Not specified but at least exceeds 0.99
E2 Low probability (<1%)	0.99
E3 Medium probability [1%,10%]	0.9
E4 High probability [10%,100%]	0.1

E. RELIABILITY MODEL

Judging whether a scheduling for a series of tasks is reliable is mainly based on two kinds of reliability: accumulated communication reliability and accumulated execution reliability [27]. Since the communication reliability has been studied in [22]–[24], while the execution reliability is rarely considered, we mainly investigate the execution reliability in our work. The execution reliability is affected by two main types of failures, namely the transient failure (i.e., random hardware failures) and permanent failure. Since the execution reliability and the random hardware failure are more closely related in some functional safety standards (e.g., IEC 61508 and ISO 26262 [9], [10], the transient failure is only considered. In addition, IEC 61508 and ISO 26262 define a concept of exposure for transient failure, which is the probability that the hazardous events may happen in the system. According to IEC 61508, the corresponding execution reliability requirement can be found in Table 1 for a given exposure level. As shown in Table 1, the exposure level and the execution reliability requirement are negatively correlated. When the exposure level is rising, the corresponding reliability requirement is going down, which increases the risk of failures in the system.

As noted in IEC 61508, the occurrence frequency of the transient failure follows a Poisson distribution during the hardware lifecycle [13]. Thus, let λ_j denotes the failure rate (constant failure rate per unit time) of the processor s_j. Then the execution reliability of subtask t_i executed on s_j is defined as

$$R_{ij} = e^{-\lambda_j \omega_i / F_j} \tag{9}$$

Leverage the model from [27], the accumulated execution reliability of the entire DAG-based task can be calculated as the product of each subtask's reliability:

$$R(G) = \prod_{t_i \in \mathbb{T}} \sum_{s_j \in \mathbb{S}} x_{ij} R_{ij} \tag{10}$$

where x_{ij} is the representation of the subtask's allocation, i.e.,

$$x_{ij} = \begin{cases} 1, & \text{if task } t_i \text{ is assigned to processor } s_j \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

IV. PROBLEM FORMULATION

In this section, we formulate an optimization problem to minimize the UE's energy consumption to execute the task G with the constraints of reliability and latency requirements.

For brevity, the energy consumption of subtask t_i in local computing and offloading computing is uniformly represented by ε_i, which is defined as follows

$$\varepsilon_i = x_{i0}\varepsilon_i^l + \sum_{j=1}^{m} x_{ij}\varepsilon_{ij}^r \tag{12}$$

Then, the problem is formulated as

$$\min \sum_{i=1}^{n} \varepsilon_i$$

Subject to :

$$L(G) \le L_{req} \tag{13}$$

$$R(G) \ge R_{req} \tag{14}$$

$$\sum_{j=0}^{m} x_{ij} = 1 \tag{15}$$

$$\tau(t_i) \ge \tau(t_k) + \sum_{j=0}^{m} x_{kj} L_{kj}, \quad \forall t_k \in pred(t_i) \tag{16}$$

$$\tau(t_i) + \sum_{j=0}^{m} x_{ij} L_{ij} \le \tau(t_k), \quad \forall t_k \in succ(t_i) \tag{17}$$

$$x_{ij} \in \{0, 1\} \tag{18}$$

The objective function is to minimize the total subtasks' energy cost, which contains the cost for execution on the UE and the cost for transmission to the MEC servers.

The constraint (13) indicates that the overall completion latency of the application G should be less than or equal to the given latency constraint. Note that $L(G)$ is not simply accumulated by the latency of each subtask because of the parallel execution. In section V, how to calculate $L(G)$ is shown. Constraint (14) indicates that the total execution reliability of the application should be greater than the given execution reliability constraint. Constraint (15) indicates each subtask can only be executed on one place, i.e., UE or one of the MEC servers. Constraint (16) and constraint (17) indicate that the successor subtasks must wait until their predecessor subtasks have been finished, where $\tau(t_i)$ is the start time of subtask t_i.

For this problem, it can be recognized as an extended bin packing problem [28], which is well-known to be NP-hard. Thus, we need a fast algorithm to solve this NP-hard problem within polynomial time.

V. A HEURISTIC ALGORITHM

The problem is decomposed into three sub-problems: prioritizing subtask, satisfying reliability and latency requirements and minimizing UE's energy consumption.

A. SUBTASK PRIORITIZING

As shown in Figure 1, there are pre- and post-dependencies between sub-tasks. Successor subtasks must wait for their predecessor subtasks to be completed before they can be assigned, while subtasks without dependencies can be executed in parallel on different processors. For example, in Figure 1, subtask t_2 needs to wait for subtask t_1 to be finished before it begins execution, while it can be executed in parallel with subtask t_3 to reduce the overall latency. Thus, figuring out the execution order of the subtasks is beneficial to improve the concurrency of the system, thereby reducing the execution latency of the subtasks.

For that, we use the subtask's upward rank value $rank_u$ [14] to measure subtasks' priority, which is defined as

$$rank_u(t_i) = \bar{L} + \max_{t_k \in succ(t_i), s_j \in \mathbb{S}, s_l \in \mathbb{S}} \{e_{ik}(s_j, s_l) + rank_u(t_k)\} \tag{19}$$

where $\bar{L}$ represents the average execution latency of subtasks on different processors in graph G. It can be calculated as:

$$\bar{L} = \frac{\sum_{i=1}^{n} \sum_{j=0}^{m} (\omega_i / F_j)}{(m+1)n} \tag{20}$$

As mentioned before, we neglect the latency for sending back the offloaded subtasks' results to the UE, thus, $e_{ik}(s_j, s_l)$ is the transmission latency from the UE to s_l, which can be computed as

$$e_{ik}(s_j, s_l) = \begin{cases} 0, & l = 0 \\ L_{kl}^r = \mu_k / v_l, & l \ne 0 \end{cases} \tag{21}$$

All subtasks are sorted in descending order by $rank_u$ before the application performing. If two tasks t_i and t_k are to be assigned and upward rank satisfy $rank_u(t_i) > rank_u(t_k)$, it can be considered that t_i may have a higher priority in the assignment than t_k. Note that if t_i is going to be executed, it needs to wait until all of its immediate predecessor subtasks have been finished, so that the constraint (16) and (17) are satisfied.

B. RELIABILITY AND LATENCY REQUIREMENT SATISFYING

(1) Reliability requirement

In order to satisfy the reliability requirement of the application G, our design strategy is as follows.

Suppose $\hat{\mathbb{T}}$ is the assigning sequence of $\mathbb{T}$, and the subtask to be assigned is $\hat{t}_i$ ($\hat{t}_i \in \hat{\mathbb{T}}$), $\{\hat{t}_1, \hat{t}_2, \cdots, \hat{t}_{i-1}\}$ is the set of subtasks that have been executed, and $\{\hat{t}_{i+1}, \hat{t}_{i+2}, \cdots, \hat{t}_n\}$ is the set of subtasks to be assigned. When assigning $\hat{t}_i$, assume its reliability value is supposed as $R(\hat{t}_i)$, then the current reliability of G can be calculated as

$$R(\hat{t}_i, G) = \prod_{k=1}^{i-1} R_a(\hat{t}_k) \times R(\hat{t}_i) \times \prod_{k=i+1}^{n} R_{ua}(\hat{t}_k) \tag{22}$$

where $R(\hat{t}_i, G)$ denotes the reliability value of G when assigning $\hat{t}_i$. $R_a(\hat{t}_k)$ is the actual reliability value that the assigned subtask $\hat{t}_k$ obtains. $R_{ua}(\hat{t}_k)$ denotes the reliability value that unassigned subtask $\hat{t}_k$ obtains.

Since the reliability value on an arbitrary processor is less than or equal to 1, to subject to the constraint (14), the following requirement must be met for any subtask $\hat{t}_i$.

$$R(\hat{t}_i, G) \geq R_{req} \tag{23}$$

Substituting Eq.(22) into (23), we have

$$\begin{aligned} R(\hat{t}_i) &\geq R_{req} \Big/ \left(\prod_{k=1}^{i-1} R_a(\hat{t}_k) \times \prod_{k=i+1}^{n} R_{ua}(\hat{t}_k) \right) \\ &\geq R_{req} \Big/ \left(\prod_{k=1}^{i-1} R_a(\hat{t}_k) \times \prod_{k=i+1}^{n} R_{ub}(\hat{t}_k) \right) \end{aligned} \tag{24}$$

where

$$R(\hat{t}_i) = \sum_{s_j \in \mathbb{S}} x_{ij} R_{ij} \tag{25}$$

$R_{ub}(\hat{t}_k)$ denotes the upper bound of reliability value that subtask $\hat{t}_k$ can obtain, namely

$$R_{ub}(\hat{t}_k) = \max_{s_j \in \mathbb{S}} \{R_{kj}\} \tag{26}$$

Theorem 1: To arbitrary subtask $\hat{t}_i$, it can always find a processor on the UE or MEC server to be assigned to satisfy Eq. (24) if $\hat{t}_i$'s predecessor subtasks are assigned to the processor that maximizes its reliability.

Proof: Firstly, we prove it on the entry subtask in $\hat{\mathbb{T}}$.

As we have known,

$$R_{ub}(\hat{t}_1) \times \prod_{k=2}^{n} R_{ub}(\hat{t}_k) \geq R_{req} \tag{27}$$

Eq. (27) must be satisfied, otherwise, we cannot find a solution to meet the R_{req}. Thus, we can assign $\hat{t}_1$ on the processor with the maximum reliability value, namely,

$$R(\hat{t}_1) = R_{ub}(\hat{t}_1) \geq R_{req} \Big/ \prod_{k=2}^{n} R_{ub}(\hat{t}_k) \tag{28}$$

It satisfies Eq.(24). Then, we extend it to an arbitrary subtask $\hat{t}_i$. Assume that $\hat{t}_{i-1}$ satisfies Eq.(24), namely,

$$R(\hat{t}_{i-1}) \geq R_{req} \Big/ \left(\prod_{k=1}^{i-2} R_a(\hat{t}_k) \times \prod_{k=i}^{n} R_{ua}(\hat{t}_k) \right) \tag{29}$$

For the $i-$ th subtask $\hat{t}_i$, we can get

$$\begin{aligned} &\prod_{k=1}^{i-2} R_a(\hat{t}_k) \times R(\hat{t}_{i-1}) \times R(\hat{t}_i) \times \prod_{k=i+1}^{n} R_{ua}(\hat{t}_k) \\ &\quad \geq \prod_{k=1}^{i-2} R_a(\hat{t}_k) \\ &\qquad \times R_{req} \Big/ \left(\prod_{k=1}^{i-2} R_a(\hat{t}_k) \times \prod_{k=i}^{n} R_{ua}(\hat{t}_k) \right) \\ &\qquad \times R(\hat{t}_i) \times \prod_{k=i+1}^{n} R_{ua}(\hat{t}_k) \\ &\quad = R_{req} \times R(\hat{t}_i) \Big/ R_{ua}(\hat{t}_i) \end{aligned} \tag{30}$$

If subtask $\hat{t}_i$ is assigned to a processor where $R(\hat{t}_i) = R_{ub}(\hat{t}_i)$, then, the Eq. (30) can be changed into

$$\begin{aligned} &\prod_{k=1}^{i-2} R_a(\hat{t}_k) \times R(\hat{t}_{i-1}) \times R(\hat{t}_i) \times \prod_{k=i+1}^{n} R_{ua}(\hat{t}_k) \\ &\quad = \prod_{k=1}^{i-2} R_a(\hat{t}_k) \times R(\hat{t}_{i-1}) \times R_{ub}(\hat{t}_i) \times \prod_{k=i+1}^{n} R_{ua}(\hat{t}_k) \\ &\quad \geq \prod_{k=1}^{i-2} R_a(\hat{t}_k) \times R(\hat{t}_{i-1}) \times R_{ua}(\hat{t}_i) \times \prod_{k=i+1}^{n} R_{ua}(\hat{t}_k) \\ &\quad \geq R_{req} \end{aligned} \tag{31}$$

Thus, $\hat{t}_i$ also satisfies Eq. (24).

To arbitrary subtask, we can find a processor to assign and meanwhile Theorem 1 is satisfied. ■

In the proposed algorithm, we adopt a greedy strategy that assuming each unassigned subtask is to be assigned to the UE or MEC server with the maximum reliability value, and then we find the available variable x_{ij} to satisfy Eq. (24) for reducing the size of the solution space.

(2) Latency requirement

We adopt the concept of the earliest start time (EST) and the latest finish time (LFT) to limit the subtask's execution time for satisfying the latency requirement of the application.

Firstly, the EST of the entry subtask t_{entry} on each processor s_j is

$$EST(t_{entry}, s_j) = 0 \tag{32}$$

Then, we can get other subtask t_i's EST on each processor s_j

$$EST(t_i, s_j) = \max_{t_k \in pred(t_i),\ s_l \in \mathbb{S}} \{EST(t_k, s_l) + \omega_k / F_l + e_{ki}(s_l, s_j)\} \tag{33}$$

The LFT of the exit subtask t_{exit} is

$$LFT(t_{exit}, s_j) = L_{req} \tag{34}$$

Meanwhile, we can deduce the other subtask t_i's LFT on each processor s_j

$$LFT(t_i, s_j) = \min_{t_k \in succ(t_i),\ s_l \in \mathbb{S}} \{LFT(t_k, s_l) - \omega_k / F_l - e_{ik}(s_j, s_l)\} \tag{35}$$

Algorithm 1 MUEECA

Input: $\mathbb{T}, \mathbb{S}, \mathbb{E}$
Output: $\{x_{ij}\}$
1: Use the subtask's upward rank value $rank_u$ to generate the sequenced set $\hat{\mathbb{T}}$
2: **for** each $\hat{t}_i \in \hat{\mathbb{T}}$ do
3 Initialize $\{x_{ij}\} \leftarrow 0$
4 $\varepsilon_i^{\min} \leftarrow \infty$
5 **For** each $s_j \in \mathbb{S}$ do
6 Set $x_{ij} = 1$
7 Update $L_{ij}^c \leftarrow \omega_i / F_j$,
  Compute $LFT(t_i, s_j)$, $EST(t_i, s_j)$
8 Update $R(\hat{t}_i) \leftarrow \sum_{j=0}^{m} x_{ij} R_{ij}$
9 **if** L_{ij}^c satisfies Eq.(36) and $R(\hat{t}_i)$ satisfies Eq.(24) then
10 Update $\varepsilon_i \leftarrow x_{i0}\varepsilon_i^l + \sum_{j=1}^{m} x_{ij}\varepsilon_{ij}^r$
11 if $\varepsilon_i < \varepsilon_i^{\min}$ then
12 Update $\varepsilon_i^{\min} \leftarrow \varepsilon_i$, $R_a(\hat{t}_i) \leftarrow R(\hat{t}_i)$
13 Reset $\{x_{ij}\} \leftarrow 0$
14 Set $x_{ij} = 1$
15 **end if**
16 **end if**
17 **end for**
18 **end for**

Then, we can get each subtask t_i's limited execution latency instead of the overall latency requirement of G.

$$\sum_{j=0}^{m} x_{ij} \left(LFT(t_i, s_j) - EST(t_i, s_j) \right) \geq L_{ij}^c = \omega_i / F_j \quad (36)$$

C. MINIMIZE THE UE'S ENERGY CONSUMPTION

In the following, an algorithm for minimizing the UE's energy consumption (MUEECA) is proposed, as shown in Algorithm 1. The core idea of MUEECA is that the overall requirements of reliability and latency are decomposed into the constraints of each subtask, and then the variables which satisfy the constraints are found with minimum energy consumption. MUEECA contain following main steps.

1) Use the subtask's upward rank value $rank_u$ to generate the sequenced set $\hat{\mathbb{T}}$.

2) Use Eq.(24) and Eq.(36) to obtain the reliability and latency constraint of each subtask on each processor in the UE or MEC server.

3) When assigning a subtask $\hat{t}_i$, find a processor with the minimum UE's energy cost and meanwhile satisfied the constraints as the assignment result of $\hat{t}_i$.

D. TIME COMPLEXITY ANALYSIS

The time complexity of MUEECA can be analyzed as follows.

Sorting $\mathbb{T}$ costs $O(n \log n)$ time. Traversing all subtasks and computing their energy cost on all processors take $O(nm)$ time. Calculating Eq.(24) and Eq.(36) require $O(n)$ time in worst case. Hence, the time complexity of the proposed algorithm is $O(n \log n + nm \times n)$, which can be approximated as $O(n^2 m)$.

VI. SIMULATION EVALUATIONS

In this section, we evaluate the performance of MUEECA through experimental simulation based on JAVA programming language and a system with an Intel i5 processor, 3.2 GHz CPU, 16GB RAM. We mainly study the impact of important parameters on the performance, including the number of subtasks, the number of MEC servers, the reliability requirement and the latency requirement.

A. SIMULATION ENVIRONMENT

We consider an MEC network in a circular area with a radius 200 m. There are {5, 10, 15, 20, 25} MEC servers uniformly deployed in the network. Consider a high-complexity application like face recognition [29], which can be split into {5, 10, 15, 20, 25} subtasks. The entire application's data size is a random number from 500KB to 600KB [26], and its required total number of processor cycles is also a random number between 1200-1500 Megacycles [30]. The computational capability F_0 of the UE's processor is 1GHz and F_j of the MEC server's processor is randomly set from 5 GHz to 10 GHz [31]. The energy cost within one time unit by the UE's processor, namely ρ, is 5 mW. The channel bandwidth W is set to 5 MHz and the noise power σ^2 is -100 dBm [31]. The transmission power P ranges from 50 mW to 100 mW randomly. The path loss factor is $\alpha = 4$, and the failure rate is a Gaussian random variable, namely $\lambda_j \sim N(0.005, 10^{-6})$, to simulate differences on devices. The reliability requirement R_{req} is set from 0.93 to 0.98 according to the references [13], [25], and the latency requirement L_{req} is set from 1.0s to 1.5s [26].

For comparison, we evaluate the performance of the proposed algorithm with the following algorithms.

1) Local execution: There is no offloading. All subtasks are executed locally.

2) Random assignment: Subtasks are randomly assigned on the UE or MEC server until all subtasks satisfy the requirements of reliability and latency.

3) Greedy algorithm: For each subtask, find the processor with minimum energy on the UE or MEC server to execute it. After all subtasks are assigned, compute the entire application's reliability and latency. If the requirements are not satisfied, then roll back to the previous subtask, select the suboptimal processor and continue to compute the requirements. Repeat the above steps until finding a solution that meets all requirements.

All experiments have been repeated 100 times with different network distribution to eliminate the effects of randomness.

B. PERFORMANCE EVALUATION

1) IMPACT OF THE NUMBER OF MEC SERVERS

To investigate the scalability, the proposed algorithm's performance is evaluated with different network scales with respect to the number of deployed MEC servers. Shown as in Figure 2, the number of MEC servers varies from 5 to 25,

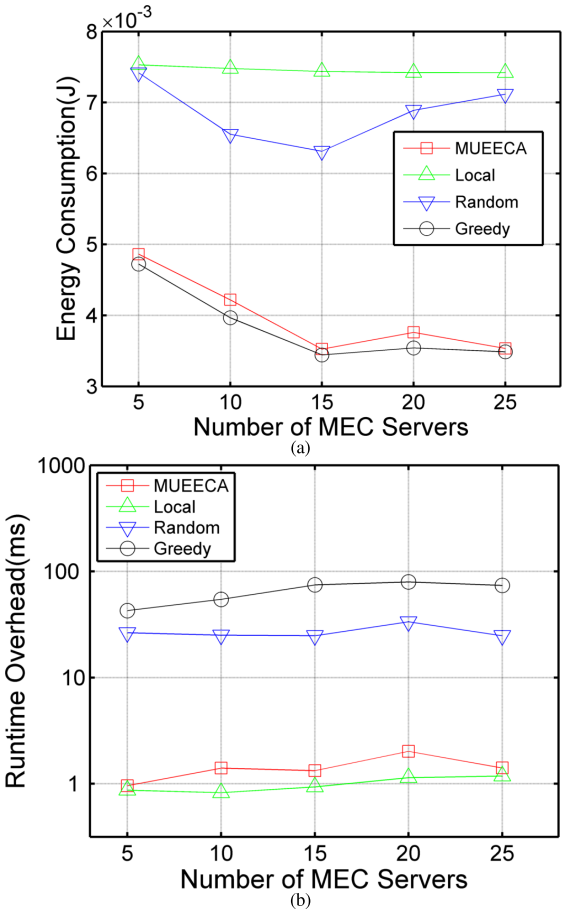


FIGURE 2. Impact of m on the subtasks' execution. (a) Energy consumption of the UE. (b) Runtime overhead.

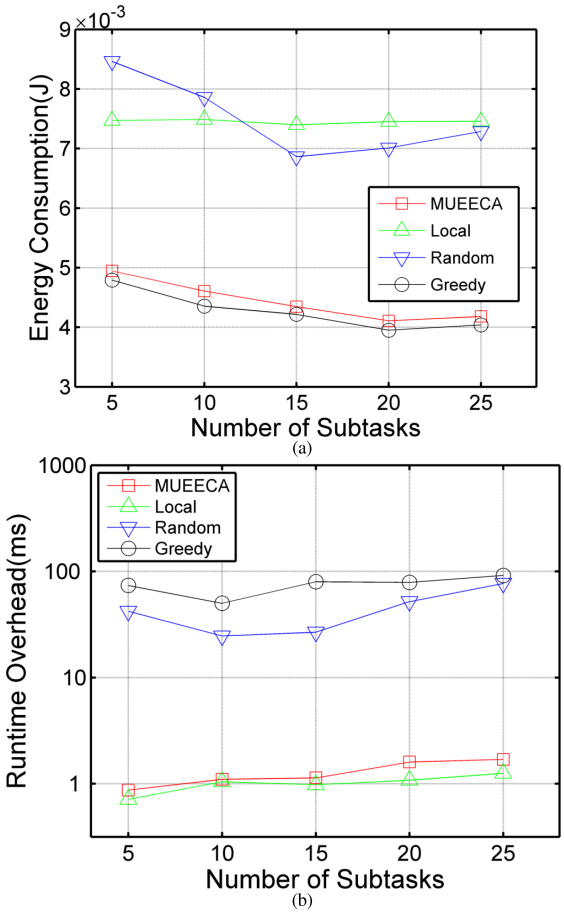


FIGURE 3. Impact of n on the subtasks' execution. (a) Energy consumption of the UE. (b) Runtime overhead.

where the number of subtasks $n = 15$, $R_{req} = 0.95$, and $L_{req} = 1500$ms. We can see that the energy consumption of the UE decreases as the number of MEC servers increases due to the increase of the solution space, where there are more options to optimize the UE's energy consumption. Moreover, the energy consumption resultant with MUEECA outperforms the local execution and the random assignment, but is slightly inferior to the greedy algorithm. Even so, the runtime overhead resultant with MUEECA is much better than the greedy algorithm. Note that the y-axis of Figure. 2(b) is a log scale. When the number of MEC servers is 25, the runtime of the proposed algorithm is close to 1 millisecond and is slightly inferior to the local execution which does not have the offloading decision process. Meanwhile, the runtime of the greedy algorithm is 73.8 ms. Note that the simulation is running on PC, if this offloading decision is running on a mobile device, it will cost more time and computational energy on the greedy algorithm. Thus, compared with the greedy algorithm, MUEECA is more suitable for the practice. The reason for this phenomenon is that MUEECA decomposes the constraints of the entire application into the sub-constraints of each subtask, and then when assigning a subtask, it can quickly find a solution satisfying the constraints. In contrast, in the greedy algorithm, it just judges whether a solution can meet the requirements in terms of reliability and latency constraints before all the decision variables $\{x_{ij}\}$ have been obtained. If the solution is not feasible, it must roll back which costs more time.

2) IMPACT OF THE NUMBER OF SUBTASKS

Figure 3 shows the impact of the number of subtasks. The number of subtasks varies from 5 to 25, where $m = 10$, $R_{req} = 0.95$, and $L_{req} = 1500$ms. We can see the energy consumption does not decrease a lot as the number of subtasks increases because the size of the entire application is almost fixed. However, the runtime of the greedy algorithm grows from 73.8ms when $n = 5$ to 91.6ms when $n = 25$, meanwhile the random assignment has similar growth too. As the number of decision variables increases, to satisfy the constraints of reliability and latency, the number of back-offs in the process of the greedy algorithm and the random assignment increases to find a feasible solution. While as shown in Figure 3(b), the runtime of MUEECA is not affected too much by the increasing number of the decision variables due to the linear solution process without any back-offs. This demonstrates the proposed algorithm is suitable for the scenarios with large solution space and multiple decision variables.

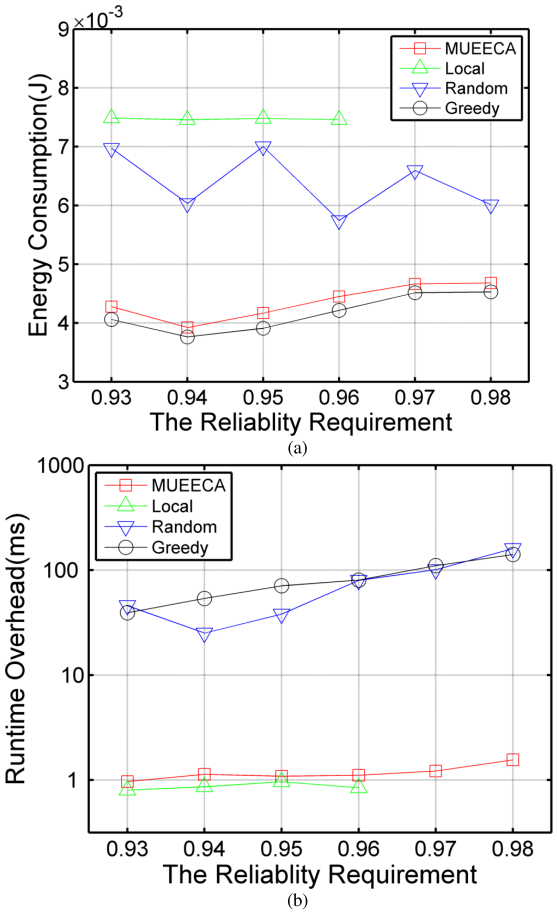


FIGURE 4. Impact of R_{req} on the subtasks' execution. (a) Energy consumption of the UE. (b) Runtime overhead.

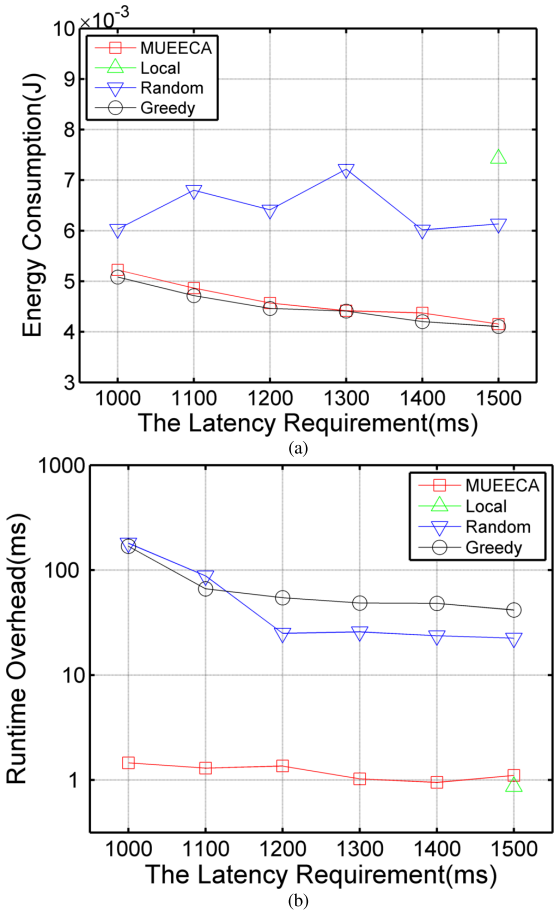


FIGURE 5. Impact of L_{req} on the subtasks' execution. (a) Energy consumption of the UE. (b) Runtime overhead.

3) IMPACT OF THE RELIABILITY REQUIREMENT

Figure 4 plots the impact of the reliability requirement where the reliability requirement varies from 0.93 to 0.98, $m = 10$, $n = 15$, and $L_{req} = 1500$ms. We can see the local execution have only 4 point data because the overall reliability of the local execution can only obtain 0.9637 at most. Thus, the local execution can't satisfy high reliability requirement when the UE has a high transient failure rate. However, with the amount of MEC devices, the UE can obtain higher reliability via the computation offloading. As shown in Figure 4(a), the energy consumption has slightly growth when the reliability requirement increasing for MUEECA and the greedy algorithm due to the constraint from Eq. (14) become stringent which makes this problem more difficult to find optimal solutions. Because of the randomness, the random assignment cannot obtain a stable result.

4) IMPACT OF THE LATENCY REQUIREMENT

Figure 5 compares the impact of the latency requirement. The latency requirement varies from 1000ms to 1500ms, where $m = 10$, $n = 15$, and $R_{req} = 0.95$. This is the case where the latency constraint becomes stringent, and an effective subtask assignment is critical to meet this deadline. Since the minimum latency of the local execution can only achieve 1505ms, it has only one point data in Figure 5. As the constraint of latency becomes loose, the energy consumption of MUEECA and the greedy algorithm is decreasing, thereby it yields a slope in Figure 5(a). When $L_{req} = 1000$ms, the runtime of the greedy algorithm is 168.7ms, while the runtime of MUEECA is just 1.4ms, whose time cost is only 0.8% of that of the greedy algorithm. This demonstrates that MUEECA has a good performance on runtime to obtain a satisfactory result on the objective of minimizing energy consumption.

VII. CONCLUSION

In this paper, we investigate a problem to minimize the UE's energy consumption while guaranteeing the application's reliability and latency requirements. The entire application is divided into DAG-based subtasks to satisfy the requirements via the computation offloading. We formulate an integer programming problem, and then propose a fast heuristic algorithm with time complexity of $O(n^2m)$. In the proposed algorithm, we split the overall reliability and latency constraints into multiple constraints for each subtask, and then find a minimum solution in each subtask's offloading decisions. Simulation results demonstrate the proposed algorithm outperforms on energy consumption when

comparing with the random assignment and the local execution, and is much better than the greedy algorithm on runtime overhead.

REFERENCES

[1] Y. Liu, A. Liu, X. Liu, and M. Ma, "A trust-based active detection for cyber-physical security in industrial environments," *IEEE Trans. Ind. Informat.*, vol. 15, no. 12, pp. 6593–6603, Dec. 2019.

[2] X. Liu, M. Zhao, A. Liu, and K. K. L. Wong, "Adjusting forwarder nodes and duty cycle using packet aggregation routing for body sensor networks," *Inf. Fusion*, vol. 53, pp. 183–195, Jan. 2020.

[3] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1657–1681, 3rd Quart., 2017.

[4] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, "Survey on multi-access edge computing for Internet of Things realization," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2961–2991, 4th Quart., 2018.

[5] X. Liu, A. Liu, T. Wang, K. Ota, M. Dong, Y. Liu, and Z. Cai, "Adaptive data and verified message disjoint security routing for gathering big data in energy harvesting networks," *J. Parallel Distrib. Comput.*, vol. 135, pp. 140–155, Jan. 2020.

[6] A. Zhou, S. Wang, B. Cheng, Z. Zheng, F. Yang, R. N. Chang, M. R. Lyu, and R. Buyya, "Cloud service reliability enhancement via virtual machine placement optimization," *IEEE Trans. Services Comput.*, vol. 10, no. 6, pp. 902–913, Nov. 2017.

[7] Y. Sharma, B. Javadi, W. Si, and D. Sun, "Reliability and energy efficiency in cloud computing systems: Survey and taxonomy," *J. Netw. Comput. Appl.*, vol. 74, pp. 66–85, Oct. 2016.

[8] Y. Alsenani, G. Crosby, and T. Velasco, "SaRa: A stochastic model to estimate reliability of edge resources in volunteer cloud," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, San Francisco, CA, USA, Jul. 2018, pp. 121–124.

[9] *ISO 26262-Road Vehicles-Functional Safety International Organization for Standardization*, ISO Standard 26262, 2011.

[10] V. Vyatkin, "Software Engineering in Industrial Automation: State-of-the-Art Review," *IEEE Trans Ind. Informat.*, vol. 9, no. 3, pp. 1234–1249, Aug. 2013, doi: 10.1109/TII.2013.2258165.

[11] X. Liu, T. Wang, W. Jia, A. Liu, and K. Chi, "Quick convex hull-based rendezvous planning for delay-harsh mobile data gathering in disjoint sensor networks," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published.

[12] X. Liu and P. Zhang, "Data drainage: A novel load balancing strategy for wireless sensor networks," *IEEE Commun. Lett.*, vol. 22, no. 1, pp. 125–128, Jan. 2018.

[13] G. Xie, G. Zeng, Y. Chen, Y. Bai, Z. Zhou, R. Li, and K. Li, "Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2017.2665552.

[14] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002, doi: 10.1109/71.993206.

[15] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through VM migration and transmission power control," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 810–819, May 2017, doi: 10.1109/TC.2016.2620469.

[16] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *Proc. IFIP Netw. Conf. (IFIP)*, May 2015, pp. 1–9.

[17] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016, doi: 10.1109/TNET.2015.2487344.

[18] S. Mu, Z. Zhong, D. Zhao, and M. Ni, "Joint job partitioning and collaborative computation offloading for Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 1046–1059, Feb. 2019, doi: 10.1109/JIOT.2018.2866945.

[19] E. Meskar, T. D. Todd, D. Zhao, and G. Karakostas, "Energy aware offloading for competing users on a shared communication channel," *IEEE Trans. Mobile Comput.*, vol. 16, no. 1, pp. 87–96, Jan. 2017, doi: 10.1109/TMC.2016.2538227.

[20] J. Ren, G. Yu, Y. Cai, and Y. He, "Latency optimization for resource allocation in mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 8, pp. 5506–5519, Aug. 2018, doi: 10.1109/TWC.2018.2845360.

[21] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018, doi: 10.1109/JSAC.2018.2815360.

[22] C.-F. Liu, M. Bennis, M. Debbah, and H. V. Poor, "Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing," *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4132–4150, Jun. 2019.

[23] J. Liu and Q. Zhang, "Offloading schemes in mobile edge computing for ultra-reliable low latency communications," *IEEE Access*, vol. 6, pp. 12825–12837, 2018.

[24] B. Han, S. Wong, C. Mannweiler, M. R. Crippa, and H. D. Schotten, "Context-awareness enhances 5G multi-access edge computing reliability," *IEEE Access*, vol. 7, pp. 21290–21299, 2019.

[25] G. Xie, Y. Chen, R. Li, and K. Li, "Hardware cost design optimization for functional safety-critical parallel applications on heterogeneous distributed embedded systems," *IEEE Trans. Ind. Informat.*, vol. 14, no. 6, pp. 2418–2431, Jun. 2018, doi: 10.1109/TII.2017.2768075.

[26] X. Lyu, H. Tian, W. Ni, Y. Zhang, P. Zhang, and R. P. Liu, "Energy-efficient admission of delay-sensitive tasks for mobile edge computing," *IEEE Trans. Commun.*, vol. 66, no. 6, pp. 2603–2616, Jun. 2018, doi: 10.1109/TCOMM.2018.2799937.

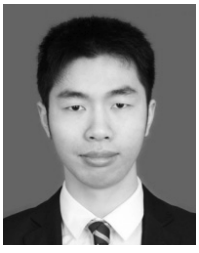
[27] L. Zhao, Y. Ren, and K. Sakurai, "Reliable workflow scheduling with less resource redundancy," *Parallel Comput.*, vol. 39, no. 10, pp. 567–585, Oct. 2013.

[28] V. V. Vazirani, *Approximation Algorithms*. New York, NY, USA: Springer, 2013.

[29] R. He, X. Wu, Z. Sun, and T. Tan, "Wasserstein CNN: Learning invariant features for NIR-VIS face recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 7, pp. 1761–1773, Jul. 2019.

[30] Y. Hao, M. Chen, L. Hu, M. S. Hossain, and A. Ghoneim, "Energy efficient task caching and offloading for mobile edge computing," *IEEE Access*, vol. 6, pp. 11365–11373, 2018, doi: 10.1109/ACCESS.2018.2805798.

[31] L. Yang, H. Zhang, M. Li, J. Guo, and H. Ji, "Mobile edge computing empowered energy efficient task offloading in 5G," *IEEE Trans. Veh. Technol.*, vol. 67, no. 7, pp. 6398–6409, Jul. 2018, doi: 10.1109/TVT.2018.2799620.



HAOLIN LIU received the B.Eng. and Ph.D. degrees from Sichuan University, Chengdu, China, in 2010 and 2015, respectively. He is currently a Lecturer with the College of Information Engineering, Xiangtan University, Xiangtan, China. His research interests include mobile edge computing, wireless sensor networks, and the Internet of Things. He is a member of CCF.

LE CAO received the B.S. degree in computer science and technology from the Hunan University of Arts and Sciences, Changde, China, in 2017. He is currently pursuing the M.S. degree in computer technology with Xiangtan University, Xiangtan, China. His research interests include mobile edge computing, network slicing, and convex optimization.

TINGRUI PEI received the B.Eng. and M.S. degrees from Xiangtan University, Xiangtan, China, in 1992 and 1998, respectively, and the Ph.D. degree in signal and information processing from the Beijing University of Posts and Telecommunications, in 2004. He is currently a Professor and a Doctoral Supervisor with Xiangtan University. He currently focuses on the research of wireless sensor networks (WSN), compressed sensing, ad hoc, mobile communication networks, and social computing. He is a member of CCF.

QINGYONG DENG (M'18) received the master's degree in signal and information processing from Xiangtan University, China, in 2009, and the Ph.D. degree from the Beijing University of Posts and Telecommunications (BUPT), China, in 2019. He is currently a Lecturer with the College of Information Engineering, Xiangtan University, China. He has published more than ten referred journal articles. His current research interests include the Internet of Things, compressed sensing, wireless networks, and 5G. He is a member of CCF.

JIANG ZHU received the M.S. and Ph.D. degrees in control science and engineering from Hunan University, Changsha, China, in 2005 and 2011, respectively. He is currently an Associate Professor with the College of Information Engineering, Xiangtan University, Xiangtan. His current research interests include intelligent information processing, mobile communication networks, and parallel distributed systems.