IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 16, 2018, accepted December 7, 2018, date of publication January 7, 2019, date of current version February 4, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2891113

Code-Partitioning Offloading Schemes in Mobile Edge Computing for Augmented Reality

JIANHUI LIU AND QI ZHANG
Department of Engineering, Aarhus University, 8000 Aarhus, Denmark
Corresponding author: Qi Zhang (qz@eng.au.dk)

ABSTRACT Augmented reality (AR) is one of the emerging use cases relying on ultra-reliable and low-latency communications (uRLLC). The AR service is composed of multiple dependent computational-intensive components. Due to the limited capability of user equipment (UE), it is difficult to meet the stringent latency and reliability requirements of AR service merely by local processing. To solve the problem, it is viable to offload parts of the AR task to the network edge, i.e., mobile edge computing (MEC), which is expected to extend the computing capability of the UE. However, MEC also incurs extra communication latency and errors on the wireless channel; therefore, it is challenging to make an optimum offload decision. So far, a little of state-of-the-art work has considered both the latency and reliability of the MEC-enabled AR service. In this paper, we study the scenario multiple edge nodes cooperate to complete the AR task. The dependency of task components is modeled by a directed acyclic graph through code partitioning. We aim to minimize the service failure probability (SFP) of the MEC-enabled AR service considering reliability and latency. We design an integer particle swarm optimization (IPSO)-based algorithm. Although the solution of IPSO-based algorithm approaches the optimum of the problem, it is infeasible to use IPSO for real-time AR services in practice due to the relatively high computational complexity. Hence, we propose a heuristic algorithm, which achieves a performance close to that of the IPSO-based algorithm with much lower complexity. Compared with state-of-the-art work, the heuristic algorithm can significantly improve the probability to fulfill the targeted SFP in various network conditions. Due to the generic characteristics, the proposed heuristic algorithm is applicable for AR services, as well as for many other use cases in uRLLC.

INDEX TERMS 5G, mobile edge computing, ultra-reliable low latency communications, augmented reality, code-partitioning offloading.

I. INTRODUCTION

Augmented reality (AR) techniques combine and interact digital content with physical reality environment in real time, which can be widely utilized in various fields, like health care, entertainment, education, and intelligent driving, etc. [1]. To this end, AR has attracted tremendous attentions from both industry and academia. So far, many advanced AR devices and platforms have emerged, such as Google Glass,[1] Microsoft HoloLens,[2] Magic Leap One[3] and so on. However, the AR service requires to complete intensive computations, e.g. camera calibration, mapping, tracking and rendering, within very short response time, where the motion-to-photo (MTP) latency is generally considered as less than 15-20 milliseconds and the packet error rate is expected below 10^{-5}, otherwise users will feel detached and nausea [2]. Therefore, AR service is one of the appealing use cases relying on ultra-reliable and low latency communications (uRLLC) in the fifth generation (5G) communication systems. Additionally, considering the limited computation capacity at user equipment (UE), it imposes challenges to meet the latency and reliability requirements of the AR service merely by local processing.

Mobile edge computing (MEC) is one of the promising technical enablers for AR service. It is a paradigm that distributes computation and storage resources at the edge of networks, such as WiFi routers and gateways at smart home, micro data center and Cloudlet between users and the core cloud. Compared with the core cloud server, the edge nodes (ENs) provide extended computational capabilities to UEs with shorter distance, which can significantly reduce the

[1]https://x.company/glass/
[2]https://www.microsoft.com/en-IE/hololens
[3]https://www.magicleap.com/

response time and energy consumption taken by service execution. However, extra communication cost, like transmission errors and latency, is incurred by offloading task to MEC through wireless channel. To fulfill the stringent latency and reliability requirements of the MEC-enabled AR, it is necessary to make optimal offloading strategy, i.e. what tasks should be offloaded, where tasks will be offloaded to, and when to execute offloading.

The naive strategy is simply to decide whether to compute the whole task locally or remotely. However, more sophisticated strategy is to offload parts of the task to ENs and compute different parts of the task in parallel to reduce latency further, if the task can be partitioned. Some work studies the partial offloading by partitioning the task into independent data bits with arbitrary granularity, of which task model is too ideal for use cases in reality. Instead, the execution of a task is normally composed of multiple methods or threads. Each method or thread is regarded as one sub-task. Parts of these sub-tasks are allowed to be offloaded and computed remotely, which is known as code-partitioning offloading [3]. For example, the execution of AR service is composed of a number of procedures including video frame capture, camera calibration, registration, tracking, 2D/3D rendering and display [4], where the raw video frame capture and virtual information display have to be completed locally, but other methods can be computed remotely. The partitioned sub-tasks are dependent on each other, of which the dependency is normally modeled by a directed acyclic graph (DAG). Furthermore, some sub-tasks may be merged or split further to obtain different partitioning granularities in terms of the performance requirements.

At present, various mobile computing platforms and algorithms are proposed for optimizing the code-partitioning offloading strategy, where the latency and energy consumption of the offloading have been well investigated in a variety of scenarios [5]–[8]. However, little attention is paid to the reliability performance. In particular, to enable AR service with MEC paradigm, the design of the code-partitioning offloading strategy faces the challenges as follows.

- It is difficult to meet the reliability and latency requirements of the MEC-enabled AR service simultaneously. The outage probability on wireless channel generally dominates the reliability. Offloading more sub-tasks to ENs may reduce the latency of completing the task, but could increase the communication error probability on wireless channel.
- The computation and communication resources at ENs are limited for a UE. Processing the sub-tasks in parallel may incur resource contentions within ENs and wireless channel, which brings negative impact on the latency of completing the AR task.
- Due to the dependency among sub-tasks, it is complex to efficiently schedule the edge resource to compute AR task at an EN. The complexity of the offloading strategy will increase further, when multiple ENs are involved.

In this paper, we study the code-partitioning offloading strategy for AR service, where multiple ENs cooperatively compute the AR task for the UE. Consider the features of MEC-enabled AR system, we systematically model the reliability and latency of completing the task, and formulate an optimization problem. In summary, the main contributions of this paper are presented as follows.

- Due to the limited resources at wireless channel and ENs, the queuing latency, as well as the communication and computation latency, is considered for modeling the latency to complete the AR task.
- To ensure the reliability of the MEC-enabled AR service, we model the service failure probability which takes into account the probability of communication error, computation failure, and timeout.
- To strike a balance between the latency and reliability, we formulate an optimization problem to minimize the service failure probability subject to the latency constraint.
- To solve the nonconvex problem, we design an integer Particle Swarm Optimization (IPSO) based algorithm and a low-complexity heuristic algorithm to optimize the offloading strategy.
- The numerical results present the probability of the proposed algorithms fulfilling a given tolerable service failure probability, which highlights the effectiveness of the proposed algorithms under various network conditions.

The rest of this paper is organized as follows. In Section II, we review the related work on computation offloading strategy and MEC-enabled AR/virtual reality (VR) system. In Section III, we present the system model and formulate an optimization problem. An IPSO-based algorithm and a heuristic algorithm are proposed to solve the formulated problem in Section IV and Section V, respectively. The simulation results are presented and discussed in Section VI. Finally, we conclude our work in Section VII.

II. RELATED WORK

MEC is a promising enabler to shorten the latency of computational-intensive applications, which has attracted much attention from both industry and academia in recent years. In 2014, European Telecommunications Standards Institute (ETSI) was set up to create industry specifications for MEC, which has been supported by Huawei, IBM, Intel, Nokia Networks, NTT DoCoMo, Vodafone, etc. [9].

In MEC paradigm, computation offloading is a key technique to allow users leveraging the computation capabilities at the network edge. According to the granularity of offloaded task, computation offloading schemes can be categorized into binary offloading and partial offloading. The binary offloading decides whether a computation task is computed at local CPU or remote server. Sardellitti *et al.* [10] jointly optimized the radio resources and the computational resources to minimize the overall users' energy consumption using binary offloading. Chen *et al.* [11] proposed a game

theoretic approach for the computation offloading decision among multiple users.

The partial offloading scheme segments a task into a set of sub-tasks and offloads parts of them to remote server. Wang *et al.* [12] investigated partial offloading by jointly optimizing the computational speed, transmit power and offloading ratio. Munoz *et al.* [13] proposed a framework for the joint optimization of the radio and computational resource usage exploiting the tradeoff between energy consumption and latency. Considering dense deployment of future networks, offloading task from one UE to multiple nearby ENs has potential to improve the performances of the system. Chiu *et al.* [14] pursued the ultra-low latency leveraging computation resource of multiple ENs. Dinh *et al.* [15] observed performance gain in energy consumption and latency when multiple ENs were considered.

Compared with the binary offloading scheme, the partial offloading scheme provides more flexible offloading strategy and utilizes the parallelism among sub-tasks execution to reduce latency. However, above work considers the model that can partition a task into arbitrary data bits, which is too ideal for use cases in reality.

To this regard, code-partitioning offloading has been well investigated to optimize the performances of the latency and energy consumption. Zhang and Wen [5] leveraged partial critical path (PCP) analysis to minimize the energy consumption while meeting a deadline in mobile cloud computing paradigm. Deng *et al.* [6] minimized the energy consumption of UE with the latency constraint of completing task by Binary Particle Swarm Optimizers (BPSO). Considering the cloud computing paradigm, Mahmoodi *et al.* [7] maximized the saved energy of task offloading with the constraint of delay, and modeled the problem as a linear optimization by replacing binary offloading indicator (at local or cloud) with a new variable, which is difficult to apply to the scenario with multiple ENs. Kao *et al.* [8] proposed a fully polynomial time approximation scheme to minimize the latency while meeting cost constraints for both deterministic and dynamic environments, which cannot be used to solve our problem directly. However, these work failed to consider that the available resource for a UE is limited in the network. If the parallel sub-tasks of a UE are utilizing the same server or wireless channel simultaneously, the resource contention occurs. In this case, the queuing latency for the contentions cannot be ignored.

The computation offloading for MEC-enabled AR or VR was recently investigated in [16]–[19]. Sun *et al.* [16] and Yang *et al.* [17] jointly optimized the computation offloading and caching policy for VR services, respectively. Al-Shuwaili and Simeone [18] minimized the energy consumption on UEs utilizing the feature that UEs can share the same content of AR service. Liu *et al.* [19] made a tradeoff between latency and accuracy of AR service by adjusting the frame resolution to offload. Liu and Zhang [20] investigated how to optimally offload the AR task in stochastic wireless environment without prior channel state information.

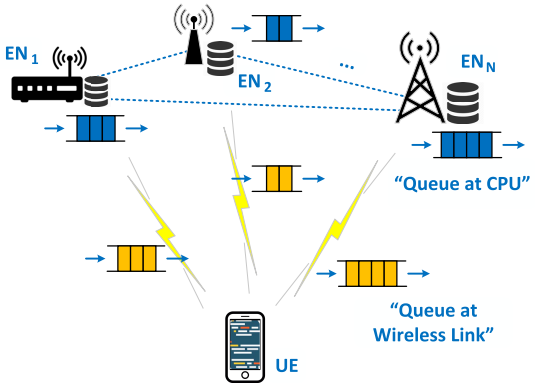

FIGURE 1. Network architecture.

Little of the state-of-the-art work on computation offloading discusses the reliability requirements. Azimi *et al.* [21] studied the transmission failure probability using superposition coding on wireless fading channel, and optimized the energy consumption with the constraints of the latency and reliability. Liu *et al.* [22] aimed to minimize the energy consumption, and modeled the latency and reliability constraints by users' task queue lengths according to the extreme value theory. It is known that task offloading consists of communication and computation. However, little of those work considers the computation reliability. Moreover, compared to the research in this paper, our preliminary work in [23] jointly optimized the reliability and latency of the offloading with a different optimization objective, which ideally partitioned the task into arbitrary bits and only took into account the bit error rate of wireless channel in the reliability modeling. It will be more realistic in this paper to optimize both communication and computation reliability of the code-partitioning offloading.

III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, the network architecture and task model are described, followed by the latency and reliability model of completing the computation task. Finally, an optimization problem is formulated.

A. ARCHITECTURE AND TASK MODEL

We consider the scenario that N ($N \geq 1$) ENs cooperatively complete the computation task for a UE illustrated in Fig. 1. The UE has been granted the channel resource and computing capacity of ENs before the AR service can start, and the offloading decision is focused on achieving the optimal reliability and latency for the UE. In the process of code-partitioning offloading, the UE can split the code of the task into V inter-dependent methods/threads, i.e. sub-tasks, and determine the offloading strategy based on available bandwidth, data size, and computing rate of CPUs, etc. [3]. According to the offloading strategy, parts of sub-tasks are offloaded via wireless channel and executed at the selected ENs. The ENs are assumed to be connected

with each others via stable communication links. Finally, the output data of the computation from the ENs will be sent back to the UE through wireless channel.

The dependency among sub-tasks is modeled by a directed acyclic graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ are the sets of vertexes and edges, respectively. The vertex i ($i \in \mathcal{V}$) denotes a sub-task, of which the weight c_i denotes the required CPU cycles to compute sub-task i. The edge $e(i, j)$ ($e \in \mathcal{E}$) denotes the invoking relation between the sub-task i and j, which is weighted by the transferred data size b_{ij} (in *bit*) from i to j. Note that not all the sub-tasks can be offloaded to ENs due to Input/Output, hardware or external constraints [24]. Here, it is mandatory to compute the first and last sub-tasks at UE, as in the AR service capturing video frame, i.e. the first sub-task, and displaying, i.e. the last sub-task, must be run locally.

It is assumed that $\mathcal{N}$ is the set of CPUs on UE and ENs, where $\mathcal{N} = \{0, 1, \ldots, N\}$ and 0 denotes the CPU on UE. We use a matrix $\boldsymbol{X}_{(N+1)\times V}$ to denote the sub-tasks allocation, where each element $x_{n,i}$ ($\forall n \in \mathcal{N}, i \in \mathcal{V}$) in $\boldsymbol{X}$ is an indicator. If sub-task i is executed at CPU n, $x_{n,i} = 1$, otherwise, $x_{n,i} = 0$. Note that a sub-task can only be allocated to one of the CPUs. In other words, $\sum_{n=0}^{N} x_{n,i} = 1$ for $\forall i \in \mathcal{V}$.

Wireless channel is considered relatively stable during the processing of each task, since the coherence time at the carrier frequency of 2.5 GHz is around 200 ms for relatively static or low-mobility scenario (2 Km/h) [25], which is long enough to complete a task. The mobility issue and its effect on the offloading performance will be left for future work.

B. LATENCY MODEL

In this subsection, the total latency for completing the task is analyzed, including computation latency, communication latency and queuing latency.

1) COMPUTATION LATENCY

We assume EN n ($1 \le n \le N$) assigns computing capacity of f_n (in *cycles/s*) to the UE. We also denote the computing rate of the CPU on UE is f_0. The latency for computing the sub-task i at the CPU n is

$$T_{n,i}^{cmp} = \frac{c_i}{f_n}, \quad n \in \mathcal{N},\ i \in \mathcal{V}. \tag{1}$$

2) COMMUNICATION LATENCY

The UE communicates with each EN via the granted wireless resource. The uplink and downlink data rates (in *bits/s*) between the UE and the EN n, i.e. r_n^{ul} and r_n^{dl}, vary with the corresponding channel qualities. Moreover, we assume a fixed data rate r_0 for each transmission between two ENs. Thus, the latency for transferring data b_{ij} from CPU m to n is

$$T_{mn,ij}^{cmm} = \begin{cases} 0, & m = n \\ b_{ij}/r_n^{ul}, & m = 0, n > 0 \\ b_{ij}/r_m^{dl}, & m > 0, n = 0 \\ b_{ij}/r_0, & otherwise. \end{cases} \tag{2}$$

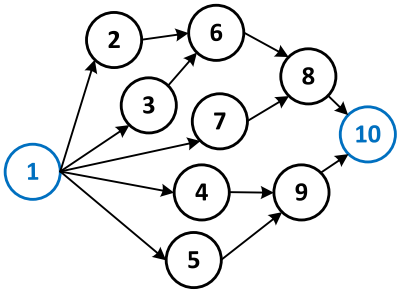

FIGURE 2. Example of task graph, where sub-tasks 1 and 10 must be computed locally and other sub-tasks can be offloaded to ENs.

3) QUEUING LATENCY

As an EN needs to serve multiple UEs. For each UE, an EN can only allocate certain amount of computing capacity and wireless resource, which may lead to the resource contentions for CPUs and wireless channel among parallel sub-tasks in the task graph. For instance, Fig. 2 shows an example of the task graph, where sub-tasks 2, 3, 4, 5 and 7 are parallel to each other. If sub-tasks 2 and 4 are assigned to the same EN, they cannot simultaneously use all the granted resource. Therefore, we assume there is a queue at each CPU and wireless link. The usages of resources follow the first-come-first-serve principle, i.e. the new arrived data cannot be processed until the previous arrived data of the queue is processed.

4) TOTAL LATENCY

Assuming τ_i^s and τ_i^f ($i \in \mathcal{V}$) are the time instant of starting and finishing computing the sub-task i, respectively, we have

$$\tau_i^f = \tau_i^s + T_i^{cmp}, \quad \forall i \in \mathcal{V}, \tag{3}$$

where $T_i^{cmp} = \sum_{n=0}^{N} x_{n,i} T_{n,i}^{cmp}$ and it denotes the latency of computing the sub-task i. τ_i^s depends on the latency of completing i's predecessors and queue length of the required computation and communication resources.

For the first sub-task, we have $\tau_1^s \equiv 0$. For the others, given sub-task i is allocated to CPU n, the CPU will not start to compute i until: i) all the input data of sub-task i has arrived at CPU n; ii) the queue at CPU n is empty. Furthermore, the input data of sub-task i from its direct predecessor p cannot be transmitted until: i) p has been completed; ii) the queue of wireless channel is empty, if it is required to transfer the input data between the UE and EN. Therefore, given $\boldsymbol{X}$, we have

$$\tau_i^s = \begin{cases} 0 & i = 1, \\ \max\left\{ q_i^{cmp}, \max\limits_{p \in \mathcal{P}_i} \left[\max\left(\tau_p^f, q_{pi}^{cmm} \right) + T_{pi}^{cmm} \right] \right\} & i > 1, \end{cases} \tag{4}$$

where q_i^{cmp} is the time instant of being available to compute sub-task i at the allocated CPU, q_{pi}^{cmm} is the time instant of being ready to transmit data b_{pi} over the assigned wireless channel, $\mathcal{P}_i$ is the set of the direct predecessors of

the sub-task i, T_{pi}^{cmm} is the latency of transmitting b_{pi} and $T_{pi}^{cmm} = \sum_{m=0}^{N}\sum_{n=0}^{N} x_{m,p}x_{n,i}T_{mn,pi}^{cmm}$.

Based on (3) and (4), the total latency of completing the task is the difference between the time instant of finishing the last sub-task, τ_V^f, and the time instant of starting the first sub-task, τ_1^s, i.e.

$$T = \tau_V^f - \tau_1^s. \tag{5}$$

Note that $\tau_1^s \equiv 0$.

C. RELIABILITY MODEL

In this subsection, we discuss the reliability of completing the task, including computation failure probability, communication error probability, timeout probability and service failure probability.

1) COMPUTATION FAILURE PROBABILITY

During the period of computing sub-tasks, the software or hardware on UE and ENs may break down, which leads to the failure of the whole task. Normally, the occurrences of the software and hardware failures are considered following Poisson process with failure rate λ^s and λ^h, respectively [26]. Therefore, the computation failure probability of computing sub-task i at CPU n is

$$F_{n,i}^{cmp} = 1 - \exp\left\{-(\lambda^s + \lambda^h)T_{n,i}^{cmp}\right\}. \tag{6}$$

2) COMMUNICATION ERROR PROBABILITY

Due to the effects of path loss, fading or shadowing, the communication reliability is significantly determined by the transmission errors on wireless channel. In this paper, we consider that the modulation and coding scheme (MCS) of device is dynamically adjusted to keep the block error rate (BLER) constant, according to channel quality. The targeted BLER of the MCS is denoted as η, and the uplink and downlink transport block (TB) size of EN n are denoted as ψ_n^{ul} and ψ_n^{dl}, respectively. Moreover, the communication errors rate between two ENs is negligible, since it is several magnitude lower than that on wireless channel. Therefore, the communication error probability of transferring data b_{ij} from CPU m to n is defined as

$$F_{mn,ij}^{cmm} = \begin{cases} 1-(1-\eta)^{b_{ij}/\psi_n^{ul}}, & m=0, n>0, \\ 1-(1-\eta)^{b_{ij}/\psi_m^{dl}}, & m>0, n=0, \\ 0, & \textit{Otherwise}. \end{cases} \tag{7}$$

Note that the retransmission is infeasible in the real-time AR service, since it may not meet the stringent latency requirement.

3) TIMEOUT PROBABILITY

It is regarded as a failure as well, if the latency of completing the task, T, is longer than the allowed delay threshold δ. Considering the variations of wireless channel and UE mobility, the latency of the same task may vary with different network conditions. The timeout probability of the task is statistically obtained as

$$F^{out} = \mathbb{P}\{T > \delta\} = \lim_{I\to+\infty}\frac{1}{I}\sum_{i=1}^{I}\mathbb{1}\left[T(i) - \delta\right], \tag{8}$$

where $T(i)$ denotes the latency of the i-th task, and

$$\mathbb{1}(x) = \begin{cases} 1, & x > 0. \\ 0, & x \le 0. \end{cases}$$

4) SERVICE FAILURE PROBABILITY

The task is failed, if the computation breaks down, communication error occurs, or the task is completed with the latency longer than the threshold. Therefore, we define the service failure probability as

$$F = 1 - \left(1 - F^{ol}\right)\left(1 - F^{out}\right), \tag{9}$$

where F^{ol} is defined as a conditional probability of offloading failure under the condition of $T \le \delta$, which is

$$F^{ol} = 1 - \prod_{i=1}^{V}\left(1 - F_i^{cmp}\right)\cdot\prod_{i=1}^{V}\prod_{j=1}^{V}\left(1 - F_{ij}^{cmm}\right)\Bigg|_{T\le\delta}, \tag{10}$$

where F_i^{cmp} is the computation failure probability of sub-task i ($F_i^{cmp} = \sum_{n=0}^{N} x_{n,i}F_{n,i}^{cmp}$), and F_{ij}^{cmm} is the communication error probability of the transferred data b_{ij} ($F_{ij}^{cmm} = \sum_{m=0}^{N}\sum_{n=0}^{N} x_{m,i}x_{n,j}F_{mn,ij}^{cmm}$). Note that, for a certain task, we define the service failure probability as

$$F = \begin{cases} F^{ol}, & T \le \delta \\ 1, & T > \delta. \end{cases} \tag{11}$$

D. PROBLEM FORMULATION

To optimize the offloading strategy of the MEC-enabled AR service, we aim to minimize the service failure probability which is an important metric for AR combining the reliability with latency. Specifically for a task, according to (11), the offloading failure probability should be minimized with the latency constraint, i.e.

$$\begin{aligned} \mathbf{P1}:\ \min_{X}\ & 1 - \prod_{i=1}^{V}\left(1 - F_i^{cmp}\right)\cdot\prod_{i=1}^{V}\prod_{j=1}^{V}\left(1 - F_{ij}^{cmm}\right), \\ s.t.\ & T \le \delta, && \text{(12a)} \\ & \sum_{n=0}^{N} x_{n,i} = 1, \quad \forall i \in \mathcal{V}, && \text{(12b)} \\ & x_{n,i} \in \{0,1\}, \quad \forall n \in \mathcal{N},\ i \in \mathcal{V}, && \text{(12c)} \\ & x_{0,1} = 1, \quad x_{0,V} = 1. && \text{(12d)} \end{aligned}$$

Obviously, (12a) is a non-convex constraint. The constraint (12b) means the sub-task i can only be executed at one of CPUs, and the constraint (12d) means the first and last sub-tasks are mandatory to be executed locally.

The formulated problem is a 0-1 integer programming problem with the non-convex constraint. Except for the first

and last sub-tasks, each sub-task is computed at one selected CPU from $(N+1)$ CPUs, the allocation of the whole task thus has $(N+1)^{V-2}$ solutions. It is obviously impossible to enumerate all the available solutions and find the optimal one due to huge complexity. Therefore, a more efficient algorithm is required for solving **P1**.

IV. OPTIMIZATION WITH IPSO-BASED ALGORITHM

As we know, the intelligent optimization techniques are widely used to search approximate solutions to non-convex problems, where particle swarm optimization (PSO) is a global optimization algorithm inspired by the social swarm behavior. Compared with other intelligent optimization techniques, like genetic algorithm, PSO is easier to implement and more efficient to converge towards the global optimum [27]. In this section, the integer PSO (IPSO) based algorithm is designed for solving the problem **P1**.

In PSO, each individual in the swarm, called a particle, denotes a potential solution to **P1**. The position and velocity of the particle k are denoted by vectors in an V-dimensional problem space, i.e. $\boldsymbol{n}_k = [n_{k1}, n_{k2}, \ldots, n_{kV}]$ and $\boldsymbol{v}_k = [v_{k1}, v_{k2}, \ldots, v_{kV}]$, respectively. Each element n_{kv} ($n_{kv} \in \mathcal{N}$) in $\boldsymbol{n}_k$ represents the CPU allocated to sub-task v. The velocity $\boldsymbol{v}_k$ determines the position of particle k in the next generation. Note that $\boldsymbol{n}_k$ is a vector in integer space, while $\boldsymbol{v}_k$ is in real space. As a result, the particle position will be obtained by rounding off the real position value to the nearest integer value, which is

$$\boldsymbol{n}_k^g = \left[\boldsymbol{n}_k^{g-1} + \boldsymbol{v}_k^{g-1}\right], \tag{13}$$

where the superscript g denotes the g-th generation of the particle ($g \geq 0$), and the operator $[\cdot]$ denotes rounding off the real value to the nearest integer.

We use $\boldsymbol{n}_k^{best}$ and $\boldsymbol{n}^{glob}$ to represent the historical best position of the particle k and the swarm best position, respectively. In each generation, the velocity of each particle is stochastically adjusted according to the $\boldsymbol{n}_k^{best}$ for the particle itself and the $\boldsymbol{n}^{glob}$, which is

$$\boldsymbol{v}_k^g = \omega \boldsymbol{k}_i^{g-1} + \phi_1\gamma_1\left(\boldsymbol{n}_k^{best} - \boldsymbol{n}_k^{g-1}\right) + \phi_2\gamma_2\left(n_k^{glob} - \boldsymbol{n}_k^{g-1}\right), \tag{14}$$

where ω is the inertia constant to control the exploration of the search space, ϕ_1 and ϕ_2 are acceleration constants, and γ_1 and γ_2 are two random numbers with uniform distribution ranging from 0 to 1. Defining the fitness function of PSO with the object function in problem **P1**, both the particle best and the swarm best positions can be derived. In this way, we can obtain the optimal reliability of computation offloading with the latency constraint through Algorithm 1.

The worst computational complexity of calculating T in each loop of Algorithm 1 is $\mathcal{O}(V^2)$. The entire complexity of IPSO-based algorithm is $\mathcal{O}(G_{num}P_{num}V^2)$, where G_{num} is the number of generations, and P_{num} denotes the number of particles. The complexity of IPSO-based algorithm is significantly lower than that of the brute-force enumeration $\mathcal{O}\left(V^2(N+1)^{V-2}\right)$. However, to obtain converged solution, we generally have $G_{num} \gg 1$ and $P_{num} \gg 1$, which leads to a high complexity. As a result, we design an algorithm with a heuristic way to decrease the complexity.

Algorithm 1 IPSO-Based Algorithm

Initialize P_{num} feasible solutions $\left[\boldsymbol{n}_1^0, \boldsymbol{n}_2^0, \ldots, \boldsymbol{n}_{P_{num}}^0\right]$ as the initial position of particles;
Calculate $F^{ol}(\boldsymbol{n}_k^0)$ for $\forall 1 \leq k \leq P_{num}$;
$\boldsymbol{n}_k^{best} \leftarrow \boldsymbol{n}_k^0$, $\boldsymbol{n}^{glob} \leftarrow \arg\min_{\boldsymbol{n}_k^0} F^{ol}(\boldsymbol{n}_k^0)$;
for $g \leftarrow 1$ to G_{num} **do**
  for $k \leftarrow 1$ to P_{num} **do**
    Calculate $\boldsymbol{v}_k^g$ and $\boldsymbol{n}_k^g$ in (14) and (13), respectively;
    Calculate T;
    if $T > \delta$ **then**
      $\boldsymbol{n}_k^g \leftarrow \boldsymbol{n}_k^{g-1}$;
    else
      Calculate $F^{ol}(\boldsymbol{n}_k^g)$
    end if
    $\boldsymbol{n}_k^{best} \leftarrow \arg\min\left\{F^{ol}(\boldsymbol{n}_k^{best}), F^{ol}(\boldsymbol{n}_k^g)\right\}$;
    $\boldsymbol{n}^{glob} \leftarrow \arg\min\left\{F^{ol}(\boldsymbol{n}^{glob}), F^{ol}(\boldsymbol{n}_k^{best})\right\}$;
  end for
end for
$F^* \leftarrow F^{ol}(\boldsymbol{n}^{glob})$
return $F^*, \boldsymbol{n}^{glob}$;

V. OPTIMIZATION WITH HEURISTIC ALGORITHM

In this section, we propose a heuristic algorithm first to solve the problem **P1** in a *tree-structured* task graph. Then we solve a more general task graph by using the proposed heuristic algorithm.

Offloading a sub-task to MEC can shorten the computation latency of the sub-task, which also reduces the corresponding computation failure probability according to (6). Meanwhile, it causes extra communication latency and the risk of communication errors over wireless channel. To make a tradeoff between the latency and reliability, the underlying rationales of the algorithm design are detailed as follows:

- **Offloading in Cluster:** It has been proven by [24] that the approach minimizing the latency for computing a sequence of sub-tasks is to simultaneously offload a set of consecutive sub-tasks to the cloud server. In other words, sub-tasks should be offloaded in cluster,[4] which is also applied to the scenario here. The reason lies in that offloading in cluster will reduce the usage of wireless channel for transferring data between sub-tasks, which will decrease the communication cost on both reliability and latency. Therefore, we will group parts of sub-tasks and offload them in clusters.

[4]A cluster is a set of sub-tasks that are connected with each other in the task graph.

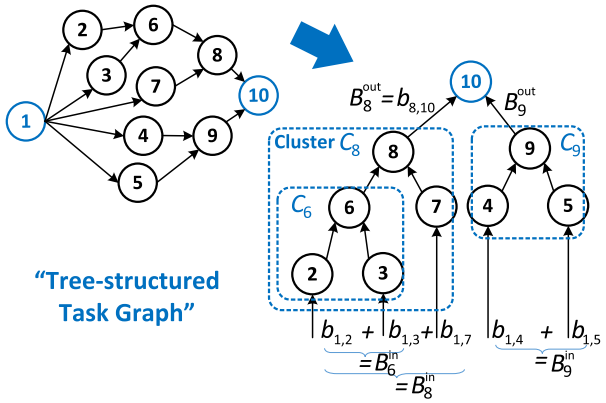

FIGURE 3. Example of tree-structured task graph and cluster definitions: the cluster $\mathcal{C}_6$ is one of sub-clusters of $\mathcal{C}_8$.

- **Latency-Guaranteed Scheduling:** On the one hand, increasing the parallelism of task computation reduces the total latency, which prefers small clusters (i.e. small number of sub-tasks or CPU cycles in one cluster). On the other hand, big cluster tends to reduce communication error probability over wireless channel due to less data transmission between the UE and EN. Considering the latency constraint, we start the algorithm from scheduling big clusters. The big cluster will be split into smaller clusters, only if the latency constraint cannot be met. The clusters will be searched in the opposite direction of sub-tasks execution.

A. TREE-STRUCTURED TASK GRAPH

In this subsection, we consider to solve the problem in a tree-structured task graph, e.g. the task graph in Fig. 3, where the *out-degree* of each vertex is equal to 1 except for that of the first and last vertexes. The direct successor of vertex i is denoted as i^+.

1) SUB-TASKS CLUSTERING

To facilitate estimating latency, we define a cluster $\mathcal{C}_k$ as a set that consists of the sub-task k and all its predecessors in the task graph except for the first sub-task. For example, in Fig. 3 we have $\mathcal{C}_8 = \{2, 3, 6, 7, 8\}$ and $\mathcal{C}_6 = \{2, 3, 6\}$. We also define cluster $\mathcal{C}_l$ is the sub-cluster of $\mathcal{C}_k$ only if $\mathcal{C}_l \subset \mathcal{C}_k$ and sub-task l is the direct predecessor of sub-task k, e.g. $\mathcal{C}_6$ is one of the sub-clusters of $\mathcal{C}_8$.

According to the design rationales discussed above, the algorithm is started with scheduling big clusters. We use $\mathcal{C}$ to denote the set of unscheduled clusters. Considering the last sub-task has to be computed locally, $\mathcal{C}$ is initialized as the set that consists of the sub-clusters of $\mathcal{C}_V$, e.g. initial $\mathcal{C} = \{\mathcal{C}_8, \mathcal{C}_9\}$ in Fig. 3. Note that the clusters in $\mathcal{C}$, e.g. $\mathcal{C}_8$ and $\mathcal{C}_9$, are disconnected with each other, as each sub-task has only one direct successor in the tree-structured task graph, except for the first and last one. In other words, each cluster in $\mathcal{C}$ can be scheduled independently.

Algorithm 2 Heuristic Algorithm for Tree Task Graph

```
Initialize C ← {C_k}, for ∀k ∈ P_V;
Initialize q̃_n ← 0, for ∀1 ≤ n ≤ N;
Calculate w_k in (15) for each C_k ∈ C
while 1 do
    C_m ← arg max_{C_k} {w_k}, C_k ∈ C;
    if δ̃_m > 0 then
        Call ClusterScheduling(C_m);
    else
        P1 has no solution, F* ← 1, X ← 0_{(N+1)×V};
        Break;
    end if
    if C is empty then
        F* ← F^{ol}(X) in (10);
        Break;
    end if
end while
return F*, X;
```

2) CLUSTER ORDERING

We assume that the edge $e(i, j)$ ($e(i, j) \in \mathcal{E}$) is the *input edge* of $\mathcal{C}_k$, if $i \notin \mathcal{C}_k$ and $j \in \mathcal{C}_k$. The sum of data at all the input edges is defined as the input data of $\mathcal{C}_k$, denoted as B_k^{in}. As shown in Fig. 3, the input data of $\mathcal{C}_8$ is $B_8^{in} = b_{1,2} + b_{1,3} + b_{1,7}$. Similarly, if $i \in \mathcal{C}_k$ and $j \notin \mathcal{C}_k$, $e(i, j)$ is defined as the *output edge* of $\mathcal{C}_k$. The data size of the output edge is defined as the output data of $\mathcal{C}_k$, denoted as B_k^{out}, e.g. $B_8^{out} = b_{8,10}$ for $\mathcal{C}_8$ in Fig. 3. Note that the input data of all the clusters all comes from the first sub-task, and each cluster only has one output edge.

Because there may be more than one clusters in $\mathcal{C}$, the scheduling order of clusters should be decided, which affects the queuing latency of sub-tasks. Obviously, we should first schedule the computation-dominant cluster which have large number of CPU cycles to compute, but small size of input/output data to transfer. Therefore, to decide the order of scheduling clusters in $\mathcal{C}$, we define the weight of the cluster $\mathcal{C}_k$ as the ratio of the sum of the required CPU cycles of $\mathcal{C}_k$, $\sum_{i \in \mathcal{C}_k} c_i$, to the sum of its input and output data, $B_k^{in} + B_k^{out}$, i.e.

$$w_k = \frac{\sum_{i \in \mathcal{C}_k} c_i}{B_k^{in} + B_k^{out}}. \tag{15}$$

The cluster with the maximal weight in the current $\mathcal{C}$ will achieve the largest offloading gain, which is thus scheduled first in each iteration as summarized in Algorithm 2.

3) CLUSTER SCHEDULING

For a given cluster $\mathcal{C}_k$, we will allocate all the sub-tasks in $\mathcal{C}_k$ to CPU n that can minimize its service failure probability within the latency constraint. However, if none of the CPU candidates can satisfy the latency constraint, we allocate sub-task k first, then $\mathcal{C}_k$ will be split into several smaller clusters which are the sub-clusters of $\mathcal{C}_k$. The sets of

Algorithm 3 Procedure of *ClusterScheduling*($\mathcal{C}_k$)

Initialized $\widetilde{F}^{ol*} \leftarrow 1$;
for $n \leftarrow 0$ to N **do**
  Calculate $\widetilde{T}_{n,k}$ in (16);
  if $T_{0,1}^{cmp} + \widetilde{T}_{n,k} + \widetilde{q}_n \leq \widetilde{\delta}_k$ **then**
    Update the elements of the n-th row in $\widetilde{\boldsymbol{X}}^{(k)}$ to 1, and other elements to 0;
    Calculate $F^{ol}(\widetilde{\boldsymbol{X}}^{(k)})$ in (10);
    if $F^{ol}(\widetilde{\boldsymbol{X}}^{(k)}) < \widetilde{F}^{ol*}$ **then**
      $\widetilde{F}^{ol*} \leftarrow F^{ol}(\widetilde{\boldsymbol{X}}^{(k)})$;
      $n^* \leftarrow n$;
    end if
  end if
end for
if $\widetilde{F}^{ol*} < 1$ **then**
  Update $\boldsymbol{X}$ with $x_{n^*,i} \leftarrow 1$, for $\forall i \in \mathcal{C}_k$;
  Update $\widetilde{q}_{n^*}$ with $\widetilde{q}_{n^*} \leftarrow \widetilde{q}_{n^*} + \widetilde{T}_{n^*,k}$;
else
  $n^* = \arg\min_n \left(T_{n,k}^{cmp} + T_{nm,ks_k}^{cmm}\right)$, $\forall n \in \mathcal{N}$;
  Update $\boldsymbol{X}$ with $x_{n^*,k} \leftarrow 1$;
  Generate and add $\mathcal{C}_p$ into $\mathcal{C}$ for $\forall p \in \mathcal{P}_k$;
  Calculate $\widetilde{\delta}_p$ in (18) for $\forall p \in \mathcal{P}_k$;
  Calculate w_p in (15), for $\forall p \in \mathcal{P}_k$
end if
Remove $\mathcal{C}_k$ from $\mathcal{C}$;
return $\boldsymbol{X}, \mathcal{C}, \widetilde{q}_{n^*}$.

unscheduled cluster $\mathcal{C}$ will be subsequently updated by removing the scheduled cluster and adding smaller new clusters. For instance, $\mathcal{C}_8$ will be removed from the initial $\mathcal{C}$ in Fig. 3, i.e. $\mathcal{C} = \{\mathcal{C}_9\}$, if sub-tasks in $\mathcal{C}_8$ are allocated. If none of the CPU candidates can satisfy the latency constraint, sub-task 8 will be allocated first, and $\mathcal{C}$ will be updated to $\mathcal{C} = \{\mathcal{C}_6, \mathcal{C}_7, \mathcal{C}_9\}$.

To estimate whether $\mathcal{C}_k$ can meet the latency constraint or not, we assume a latency constraint $\widetilde{\delta}_k$ as the latest time instant to complete transmitting the output data of $\mathcal{C}_k$, as the example shown in Fig. 4. It means, if $\mathcal{C}_k$ is assigned to CPU n, the sum of the following latency should not exceed $\widetilde{\delta}_k$: i) latency of computing the first sub-task, i.e. $T_{0,1}^{cmp}$; ii) latency of completing $\mathcal{C}_k$, i.e. $\widetilde{T}_{n,k}$; iii) the relevant queuing latency at CPU n and at the uplink and downlink, i.e. $\widetilde{q}_n$.

Specifically, completing the cluster $\mathcal{C}_k$ includes computing its sub-tasks at CPU n and transmitting its input and output data, of which latency is

$$\widetilde{T}_{n,k} = \overbrace{\sum_{i\in\mathcal{C}_k} T_{n,i}^{cmp}}^{\text{comp. sub-tasks}} + \overbrace{\sum_{i\in\mathcal{C}_k} T_{0n,1i}^{cmm}}^{\text{trans. input}} + \overbrace{T_{nm,kk^+}^{cmm}}^{\text{trans. output}}, \quad (16)$$

where k^+ denotes the direct successor of sub-task k, m denotes the CPU to compute sub-task k^+.

Regarding the queuing latency $\widetilde{q}_n (1 \leq n \leq N)$, to simplify the algorithm, we initialize $\widetilde{q}_n = 0$, and calculate $\widetilde{q}_n$ by adding

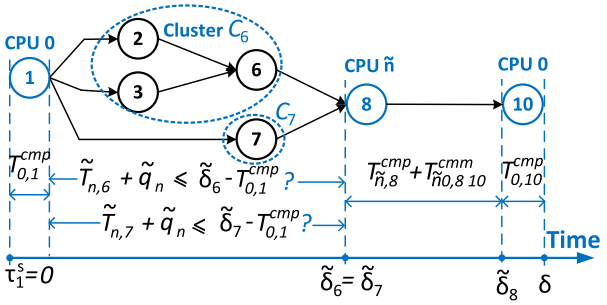

FIGURE 4. Example of cluster scheduling: the blue sub-tasks have been allocated.

the latency of processing clusters that are previously assigned to CPU n. For instance, if $\mathcal{C}_8$ has been assigned to CPU 1, the $\widetilde{q}_1$ will be updated to $\widetilde{q}_1 = \widetilde{q}_1 + \widetilde{T}_{1,8}$ which is used for estimating the queuing latency of the next cluster allocated to CPU 1.

In this way, we can check that if allocating the cluster $\mathcal{C}_k$ to CPU n can meet the latency constraint $\widetilde{\delta}_k$ or not. All the CPUs that satisfies $T_{0,1}^{cmp} + \widetilde{T}_{n,k} + \widetilde{q}_n \leq \widetilde{\delta}_k$ will be found, where the one minimizing the service failure probability is the optimal allocation for $\mathcal{C}_k$. If $T_{0,1}^{cmp} + \widetilde{T}_{n,k} + \widetilde{q}_n > \widetilde{\delta}_k$ for all the CPU candidates, we allocate the sub-task k in $\mathcal{C}_k$ only to the CPU that minimizes sub-task k's processing latency, i.e.

$$\widetilde{n} = \arg\min_n \left(T_{n,k}^{cmp} + T_{nm,kk^+}^{cmm}\right), \quad (17)$$

where k^+ denotes the direct successor of sub-task k, m denotes the CPU to compute sub-task k^+. After that, $\mathcal{C}_k$ will be removed from $\mathcal{C}$. The newly split sub-clusters of $\mathcal{C}_k$ will be added into $\mathcal{C}$. As the sub-task k has been assigned to CPU $\widetilde{n}$, the latency constraint of each $\mathcal{C}_k$'s sub-cluster $\mathcal{C}_p$ can be obtained by

$$\widetilde{\delta}_p = \widetilde{\delta}_k - \left(T_{\widetilde{n},k}^{cmp} + T_{\widetilde{n}m,kk^+}^{cmm}\right). \quad (18)$$

For clarity, Fig. 4 provides an example to illustrate the relationship among clusters' latency constraints, $\widetilde{\delta}_k$, and latency of completing clusters, $\widetilde{T}_{n,k}$, at time axis.

The algorithm ends when all the sub-tasks in the task graph are allocated. Moreover, if the case that $\widetilde{\delta}_k \leq 0$ occurs, the problem **P1** is considered as no solution, which means $F = 1$. The procedure of cluster allocation is summarized in Algorithm 3, where $\widetilde{\boldsymbol{X}}^{(k)}$ is the matrix to indicate the allocation of sub-tasks in the sub-cluster $\mathcal{C}_k$. Finally, the UE will process the AR task following the optimal solution, i.e. $\boldsymbol{X}$, obtained by the heuristic algorithm.

We provide the complexity analysis for the proposed heuristic algorithm as follows. We assume the maximal number of the direct predecessors of a sub-task in task graph is L, which results in the complexity of $\mathcal{O}(L)$ to calculate the sub-clusters' weights in initial $\mathcal{C}$. The maximal number of the loop iterations is $(V-2)$ in Algorithm 1. For each iteration, it takes L times to find the sub-cluster with the maximal weight. After that, *ClusterScheduling*($\mathcal{C}_k$) in

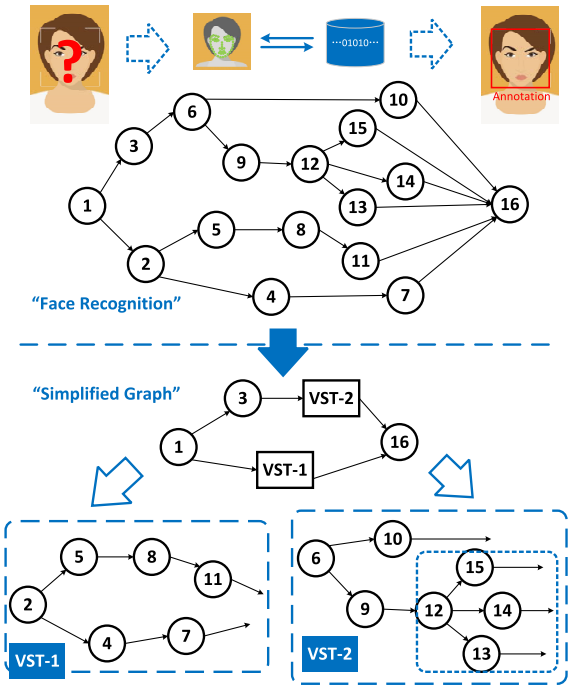

FIGURE 5. Task graph of face recognition and its simplified tree-structured graph.

Algorithm 2 is called, with the complexity $\mathcal{O}((N+1)(V-2))$. Finally, the overall complexity of Algorithm 1 is $\mathcal{O}(L) + \mathcal{O}((V-2)(L+(N+1)(V-2)))$. Due to $L \leq V-2$, the complexity of the heuristic algorithm in the worst case is $\mathcal{O}\left((N+1)(V-2)^2\right)$, which is much lower than that of IPSO-based algorithm when $G_{num} \gg 1$ and $P_{num} \gg 1$, i.e. $G_{num}P_{num} \gg (N+1)$.

B. A GENERAL TASK GRAPH

In this subsection, we describe how Algorithm 2 can be applied to solve the problem **P1** in a general task graph, where each vertex may have more than one direct successors.

Based on the observation in [8], the task graphs of many applications can be decomposed into several tree-structured graphs which start from a unique sub-task and split into multiple parallel sub-tasks. To this regard, the original task graph can be simplified into a tree-structured graph via merging the sub-task with its direct successors. For example, the task graph of a face recognition application [28] in Fig. 5 consists of two parallel sub graphs $\{2, 4, 5, 7, 8, 11\}$ and $\{6, 9, 10, 12, 13, 14, 15\}$. A simplified tree-structured task graph can be obtained when the sub-tasks in these two sub graphs are merged into two *virtual sub-tasks* (VSTs), respectively. Furthermore, we notice that the VST-1 $\{2, 4, 5, 7, 8, 11\}$ can be regarded as a tree structured graph as well, since the out-degree of sub-tasks (except for the sub-task 2) in VST-1 is one. Regarding the VST-2 in Fig. 5, it is composed of a tree graph $\{12, 13, 14, 15\}$ and other *one-out-degree* sub-tasks, which can be simplified into a tree-structured graph following the above-mentioned steps.

Therefore, we can solve the problem **P1** in a general task graph by scheduling the simplified task graph with Algorithm 2. If it cannot meet the latency constraint by scheduling the simplified graph, considering the two rationales at the beginning of this section, we continue to schedule the task graph of VSTs in the above-mentioned way, until the latency constraint can be satisfied or there is no VST in the graph any more. For instance, in Fig. 5, we schedule the simplified graph using Algorithm 2 at first. If the latency of allocation results T is larger than the latency constraint δ, i.e. $T - \delta > 0$, we can continue to schedule the task graph of the VST, e.g. VST-1, and re-schedule the sub-tasks in VST-1 using Algorithm 2. The reason lies in that, at the first round of schedule, the sub-tasks in VST-1 are allocated to the same CPU. If we re-allocate the parallel sub-tasks in VST-1 to different CPUs at the next round, the latency may be decreased, due to the increasing of the computation parallelism.

To re-schedule the sub-tasks in the VST, the latency constraint of the VST should be redefined. We assume that, at the previous round, the VST J and its direct successor J^+ are allocated to the CPU n and CPU m, respectively. Note that, at the current schedule round, the first sub-task of the VST J, e.g. sub-task 2 in VST-1 of Fig. 5, is mandatory to be executed at CPU n. The latency constraint of the VST J to re-schedule is defined as

$$\delta_J = \underbrace{\sum_{i \in \mathcal{J}} \left(T^{cmp}_{n,i} + T^{cmm}_{nm,iJ^+} \right)}_{\text{Current latency of VST}} - \underbrace{(T - \delta)}_{\substack{\text{Exceeded} \\ \text{latency}}}, \tag{19}$$

where $T - \delta > 0$. The first item of (19) means the latency of processing the VST J with the allocation of the previous round, and the second item $T - \delta$ means the exceeded latency of the current allocation to the latency constraint for the whole task.

Moreover, to handle the allocation of the general task graph, Algorithm 2 has to make a modification of its terminal condition. Instead of terminating the algorithm with any $\widetilde{\delta}_m > 0$, the main loop in the Algorithm 2 will not break until all of the sub-tasks are allocated. In this way, the latency constraint of the VST can be calculated via (19).

VI. NUMERICAL RESULTS AND DISCUSSIONS

In this section, the performances of the proposed algorithms are analyzed. The effects of the system environment parameters on the service failure probability (SFP) are studied first in a tree-structured task graph. Sequentially, we analyze how the characteristics of the task graph affect the algorithm performances. Finally, we take one of important components in AR service as an example, i.e. the face recognition application in Fig. 5, to evaluate the algorithm performances in the general task graph.

The configurations of the transmission links are based on the LTE system [29]. The SNR over wireless channel is obtained by a uniform distribution, ranging from 0 dB to

TABLE 1. Default simulation parameters.

Parameter	Value
Uplink/downlink bandwidth of each EN	10MHz
SNR over wireless link $[\mathrm{SNR}_{\min}, \mathrm{SNR}_{\max}]$	$[0, 30]$ dB
Data rate between ENs	1 Gbps
Computing rate at UE	1×10^9cycles/s [15]
Computing rate at EN $[f_{\min}, f_{\max}]$	$[4, 8] \times 10^9$cycles/s [15]
Software failure rate	8×10^{-4} s^{-1} [26]
Hardware failure rate	4×10^{-4} s^{-1} [26]

30 dB. At different SNRs, the MCS is dynamically adjusted according to the targeted BLER. The SNR-MCS mapping is achieved by the link abstraction model proposed by [30]. Various MCSs are considered combining different modulations (QPSK, 16QAM, 64QAM) and varying coding rates ranging from 1/9 to 9/10.

The task graph used in the simulation is randomly generated [31], which is determined by the main characteristics as follows. i) The number of the vertexes V in the graph. ii) Out-degree of each vertex except for the first and last vertexes, which is set as 1 for the tree structured graph. iii) Graph shape, which can be defined as the average width-to-height ratio of the graph and denoted by β. It is a characteristic that presents the parallelism degree of the task call graph. If $\beta \gg 1$,the generated task graph has high parallelism; if $\beta \ll 1$, low-parallelism graph is obtained. iv) Communication to computation ratio (CCR), which is to show the computation intensity of the task [32]:

$$\mathrm{CCR} = \frac{\sum_{(i,j)\in\mathcal{E}} b_{ij} / (\bar{r} \times |\mathcal{E}|)}{\sum_{i\in\mathcal{V}} c_i / (f_0 \times V)}, \tag{20}$$

where $\bar{r}$ denotes the average data rate over wireless channel between UE and ENs, and $|\mathcal{E}|$ is the number of edges in the graph. We assume the required number of CPU cycles for each sub-task are selected from a Gaussian distribution with mean value of 50×10^6 cycles and variance of 100 [33]. The computational intensive task will have a very low CCR.

We summarize the default simulation parameters in Table 1. The numerical results in this section are based on 5000 Monte Carlo simulations. We compare the performance of IPSO-based algorithm and Heuristic algorithm against 1) a naive offloading scheme, i.e. Full Offloading (FO) scheme, which computes all the offloadable sub-tasks at one CPU that minimizes SFP; 2) one of the representative existing algorithms, i.e. the revised partial critical path (PCP) based scheme in [5], which fails to consider the computing resource contention of parallel sub-tasks of an EN. The PCP-based scheme estimates the possible starting instant of each sub-task and add the sub-tasks with the latest starting instant into the critical path. Then the object function of sub-tasks on the critical path is greedily minimized. The number of particles and generations in IPSO-based algorithm is set to $P_{num} = 20$ and $G_{num} = 100$, respectively.

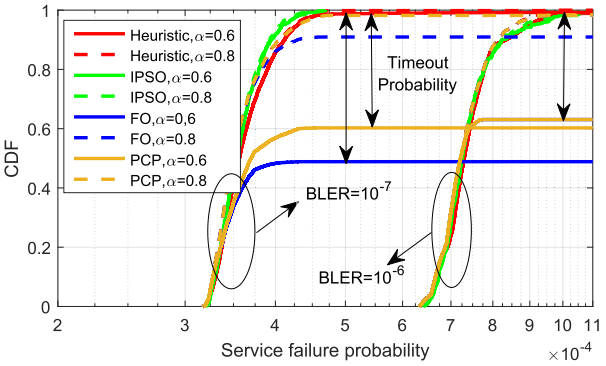

FIGURE 6. Impact of targeted BLER on service failure probability.

A. IMPACT OF TARGETED BLER

Fig. 6 shows the cumulative distribution function (CDF) curves of SFP over samples with random computing rates of ENs and SNR on wireless channel. In each simulation, the same task graph with 20 vertexes is used and the number of ENs is 3. We define $\alpha(\alpha > 0)$ as the ratio of the latency constraint to the latency of computing the task at UE. The larger α means to allow the task to be completed in a longer time. Note that the latency constraint in problem **P1** cannot be met, i.e. service timeout occurs, if i) the algorithm provides non-feasible solution for problem **P1**; ii) the algorithm cannot provide a solution, but the feasible solution of problem **P1** exists; iii) problem **P1** has no solution. For those cases, the service could be regarded as timeout, which means $F = 1$ according to the definition in (11).

From the figure, we can obtain the probability of an algorithm fulfilling the targeted SFP. For instance, when BLER[5] is 10^{-7} and $\alpha = 0.6$, with the targeted SFP increasing, the probability of all the schemes fulfilling the target rises first. Using Full Offloading scheme, about 48% of the cases can achieve the targeted SFP of 4×10^{-4}, which cannot rise any more even though we continue to increase the tolerable SFP. The reason is that the scheme always fails to provide feasible solutions to meet the latency constraint in some cases. It means the minimal timeout probability of Full Offloading scheme is 52%. The PCP-based scheme can fulfill the targeted SFP of 4×10^{-4} with probability of about 60%, since it ignores the computing resource contention of the parallel sus-tasks, and models the latency imprecisely. Compared with that, Heuristic algorithm has 98% probability to obtain SFP below 5×10^{-4}, which is significantly better than that of Full Offloading scheme and PCP-based scheme. The difference of performance between IPSO-based algorithm and Heuristic algorithm is trivial. However, it is not feasible to use the IPSO-based algorithm in real-time process, since its complexity is much higher than that of Heuristic algorithm (approximate 500:1) in terms of the aforementioned

[5]According to the 3GPP standard [34], the targeted BLER is normally defined as 0.1, which cannot fulfill the requirement of packet error rate of AR service (at level of 10^{-5}), thus the BLER of 10^{-7} or 10^{-6} is selected in our simulation

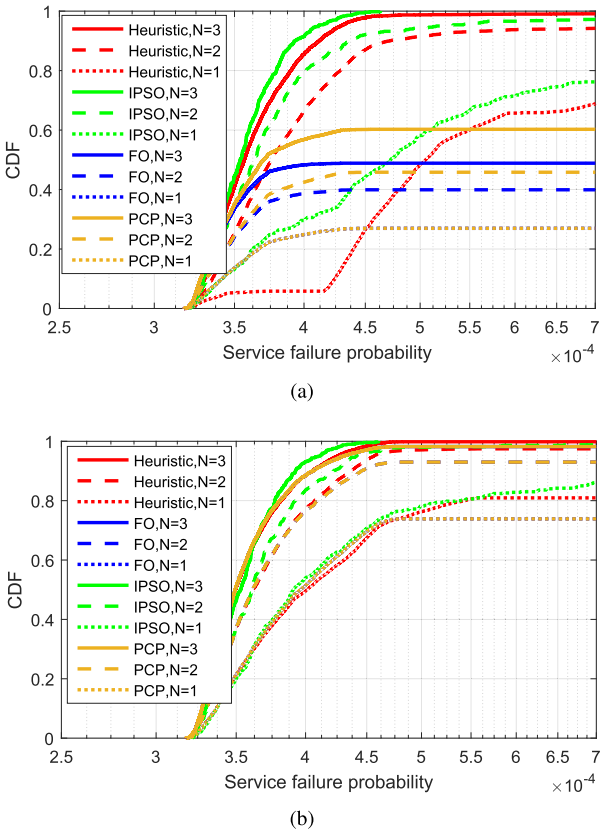

FIGURE 7. Impact of the number of ENs on service failure probability in terms of latency constraint: (a) $\alpha = 0.6$; (b) $\alpha = 0.8$.

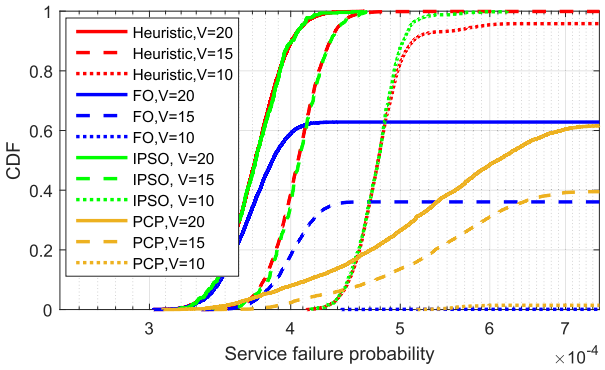

FIGURE 8. Impact of the number of sub-tasks on service failure probability.

configurations. The task is actually failed if the offloading decision cannot be made within the delay constraint.

When the targeted BLER is 10^{-6}, the lowest SFP that the schemes can obtain rises to 6.4×10^{-4}, since in this scenario the communication error probability dominates the SFP. However, with larger targeted BLER, higher order MCS can be used on wireless channel to improve the data rate, which contributes to shorten the communication latency. As a result, if the targeted SFP is larger than 8×10^{-4}, the probability of Full Offloading scheme fulfilling the target rises to around 60%, which has the same performance as the PCP-based scheme. Heuristic algorithm and IPSO-based algorithm have similar performance, which can achieve SFP below 10^{-3} with 99% probability.

If we relax the latency constraint from $\alpha = 0.6$ to $\alpha = 0.8$, the performance of all the schemes is improved, but the differences of performance between the proposed algorithms and Full Offloading schemes are reduced. It means that the proposed algorithms are more effective in the case that latency constraint is more stringent.

B. IMPACT OF THE NUMBER OF ENS

Fig. 7 is presented to show the impact of the number of ENs in the network on the SFP performance. The targeted BLER is set to 10^{-7}. Fig. 7a and 7b are simulated in terms of the latency constraint $\alpha = 0.6$ and $\alpha = 0.8$, respectively. In each simulation, the same tree-structured task graph are used. As shown in Fig. 7, the SFP performance is improved with the number of ENs increasing. In Fig. 7a, when $\alpha = 0.6$, the performance of IPSO-based algorithm is slightly better than that of Heuristic algorithm. Both algorithms can achieve more than 99% probability to fulfill SFP below 6×10^{-4} for $N = 3$. However, using Full Offloading scheme and PCP-based scheme, about 50% and 60% of the cases can fulfill the targeted SFP of 5×10^{-4}, respectively, even though the UE is allowed to access 3 ENs.

In Fig. 7b, the latency constraint ratio is relaxed from 0.6 to 0.8. The probabilities of all the schemes fulfilling the targeted SFP are improved. Heuristic algorithm has similar performance as IPSO-based algorithm. They can fulfill SFP below 5×10^{-4} with more than 98% probability for the case $N = 3$ and $N = 2$, of which the performance gain to Full Offloading scheme and PCP-based scheme is less than 5%.

C. IMPACT OF THE NUMBER OF SUB-TASKS

In Fig. 8, we illustrate the CDF curves to present how the number of sub-tasks affects the SFP performance. In each simulation, a tree-structured task graph is generated randomly for a given β and CCR. The random graphs with different number of vertexes have the same total required CPU cycles, which means that it takes the same time to locally complete the tasks with different number of sub-tasks V. The latency constraint ratio α is set to 0.6, and the number of EN candidates is 3. Each EN has the same computing rate and data rate on wireless channel, where $f_n = 5 \times 10^9$cycles/s, and $r_n^{ul} = r_n^{dl} = 25$Mbps for $\forall 1 \leq n \leq N$. From the figure, the reliability performances of the schemes are improved with the number of the sub-tasks growing. The reason lies that the CPU cycles required by the first and last sub-tasks decrease with the number of sub-tasks growing, when the total CPU cycles of the task is fixed. In other words, more computation load can be offloaded to the edge. Full Offloading scheme will have higher probability to complete the task within the latency constraint via computation offloading. Moreover, the parallelism among sub-tasks may increase

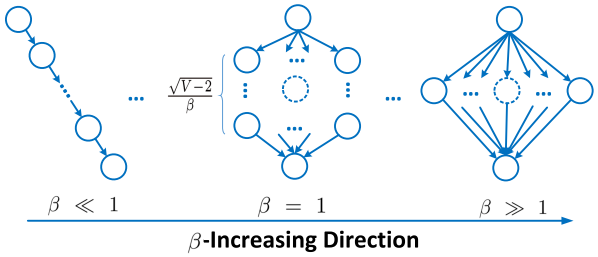

FIGURE 9. Simplified diagram of graphs with different shapes: when β is 1, the average layer of the generated graph is $\frac{\sqrt{V-2}}{\beta}$.

as well, which tends to shorten the computation latency. We can observe that the Heuristic algorithm and IPSO-based algorithm have similar performance when $V = 20$ and 15, where they have more than 99% probability to fulfill SFP below 4.5×10^{-4} and 5×10^{-4}, respectively. If $V = 10$, the probability of IPSO-based algorithm fulfilling 5.4×10^{-4} is slightly higher than that of Heuristic algorithm, at the price of higher complexity. For the PCP-based scheme, its SFP performance highly depends on the critical path selection, which is determined by the latency estimation. Since PCP-based scheme fails to consider the limited computing resource at ENs, the errors of the latency estimation lead to even worse SFP performance than that of Full Offloading scheme when the task graph varies. Furthermore, the feasible solutions of Full Offloading scheme and PCP-based scheme can be rarely obtained to fulfill any targeted SFP for $V = 10$.

D. IMPACT OF THE TASK GRAPH SHAPE

Fig. 10 presents the CDF curves of the schemes with different shapes of task graphs. All the random samples of the tree-structured task graphs over simulations have the same number of vertexes and CCR. The settings of ENs in the network are the same as the last subsection. The latency constraint ratios α in Fig. 10a and 10b are 0.6 and 0.8, respectively. In both cases, the SFP performances of all the schemes are improved with β decreasing. It is because that the decrease of β leads to out-degree of the first sub-task and in-degree of the last sub-task decreasing, as shown in Fig. 9. It means the parallelism of the task graph decreases, and less data tends to be transmitted on wireless channel, which results in lower communication error probability. Meanwhile, the queuing latency on wireless channel may reduce as well. As a result, in Fig. 10a, Heuristic algorithm can achieve SFP below 4×10^{-4} with almost 100% probability for the case $\beta = 0.25$, while it obtains SFP below 5×10^{-4} with probability less than 98% for the case $\beta = 1$. The IPSO-based algorithm has similar performance of reliability as Heuristic algorithm. Compared with that, when the β is 0.5, Full Offloading scheme only has around 60% probability to obtain SFP below 4.5×10^{-4}, which cannot rise any more with the targeted SFP increasing, while PCP-based scheme has no more than 50% probability to obtain targeted SFP of 7×10^{-4}. For the case $\beta = 1$, Full Offloading scheme and PCP-based scheme rarely have feasible solution to the problem.

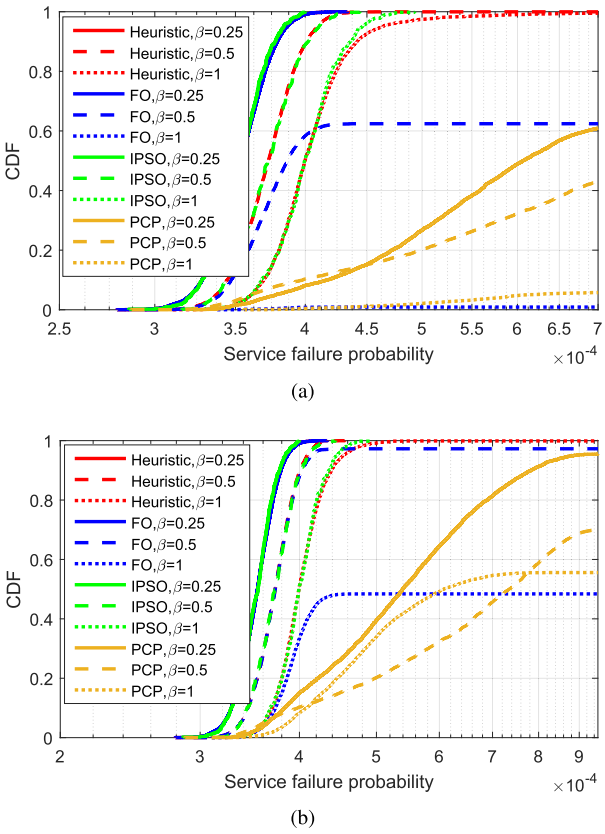

FIGURE 10. Impact of shape of task graph on service failure probability in terms of latency constraint: (a) $\alpha = 0.6$; (b) $\alpha = 0.8$.

When the latency constraint is relaxed to $\alpha = 0.8$, for all the algorithms the maximal probability fulfilling a certain targeted SFP increases.

E. IMPACT OF COMMUNICATION TO COMPUTATION RATIO

The impact of communication to computation ratio (CCR) on SFP performance is shown in Fig. 11. For a given β and the number of sub-tasks, the task graph in each simulation is randomly generated. The latency constraint ratios α in Fig. 11a and 11b are 0.6 and 0.8, respectively. According to the definition, the task graph with a smaller CCR means the task is more computationally intensive. To compute the task with the given CPU cycles, the smaller CCR is, the less data is required to be transferred, which may lead to low communication error probability on wireless channel. As a result, the SFP performance is improved with the decreasing of CCR. Specifically, as shown in Fig. 11a, Heuristic algorithm and IPSO-based algorithm have similar performance when CCR is equal to 0.5 and 1. For the case CCR $= 2$, Heuristic algorithm has 80% probability to fulfill SFP below 5×10^{-4}, while the probability of IPSO-based algorithm achieves 99%. Compared with that, Full Offloading cannot solve the problem **P1** at all when the computation intensity is low (CCR $= 2$). It can achieve SFP below 4.5×10^{-4}

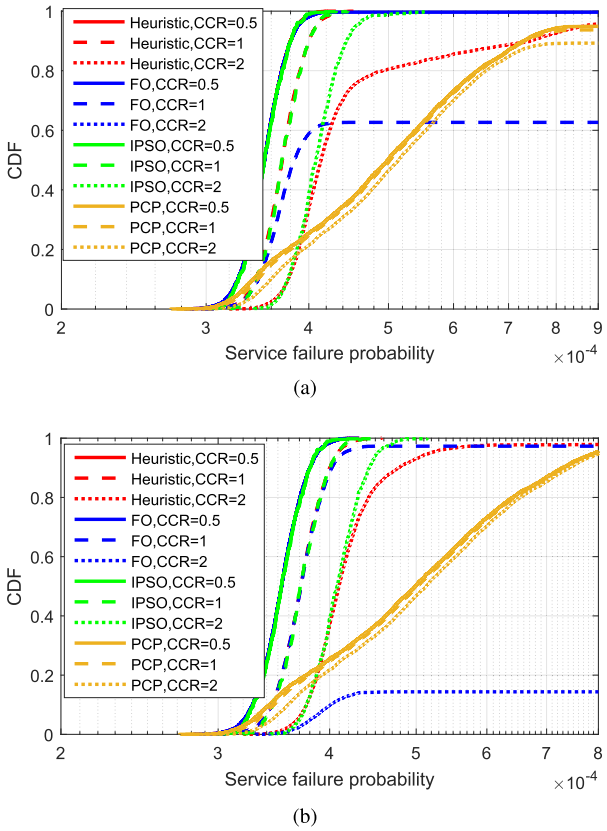

FIGURE 11. Impact of CCR of task graph on service failure probability in terms of latency constraint: (a) $\alpha = 0.6$; (b) $\alpha = 0.8$.

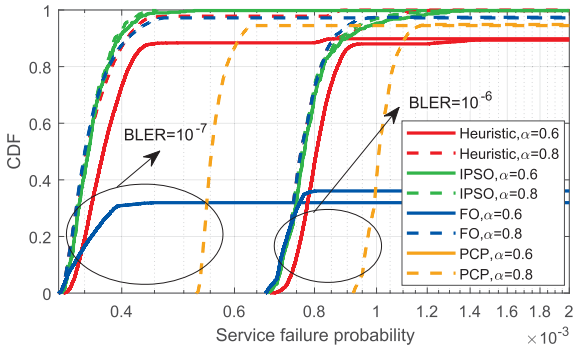

FIGURE 12. Performance comparisons in a face recognition application with various targeted BLER.

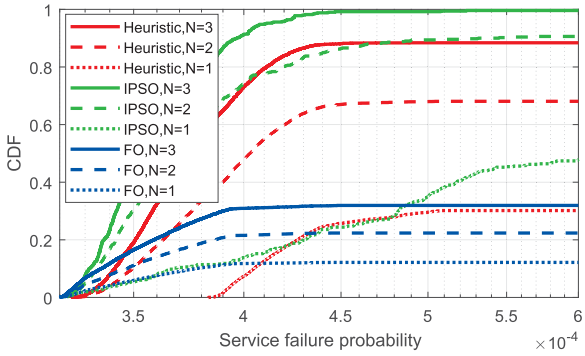

FIGURE 13. Performance comparisons in a face recognition application with various number of ENs.

with 60% probability, when the CCR is 1. The variation of CCR has little impact on the SFP of PCP-based scheme.

For $\alpha = 0.8$, the timeout probabilities of all the schemes are lower than that in the case $\alpha = 0.6$. The probability of Full Offloading scheme fulfilling the targeted 4.5×10^{-4} SFP rises to about 98% for CCR $= 1$. However, for CCR $= 2$, Full Offloading scheme can only fulfill SFP below 4.5×10^{-4} with the probability lower than 20%. Moreover, compared with the case $\alpha = 0.6$, the probability of Heuristic algorithm fulfilling 5×10^{-4} SFP rises to 90% when $\alpha = 0.8$ and CCR $= 2$.

F. PERFORMANCE COMPARISONS IN FACE RECOGNITION

In this subsection, the task graph of face recognition application in Fig. 5 is taken as an example to evaluate the performances of the proposed algorithms.

Fig. 12 compares CDF curves of SFP among schemes in terms of different targeted BLER and latency constraints. Each curve is based on 5000 samples with random EN settings. The UE is allowed to access 3 ENs simultaneously. When the required latency is $\alpha = 0.6$, SFP performance of Heuristic algorithm is significantly better than that of Full Offloading scheme. For the case the targeted BLER is 10^{-7}, Full Offloading scheme can fulfill the targeted SFP of 4×10^{-4} with about 30% probability, while the probability of Heuristic algorithm fulfilling 5×10^{-4} SFP is around 90%. If the targeted BLER reduces to 10^{-6}, the minimal timeout probability of above three schemes does not have obvious improvement. Note that the solid curve of PCP-based scheme for $\alpha = 0.6$ could hardly be shown in the figure. It means that PCP-based scheme cannot provide feasible solution to problem **P1** at all when the latency constraint is $\alpha = 0.6$.

When the latency constraint is relaxed from $\alpha = 0.6$ to $\alpha = 0.8$. The reliability of both Heuristic algorithm and Full Offloading scheme is improved and approximate to that of IPSO-based algorithm. PCP-based scheme fulfills SFP below 1.2×10^{-3} with more than 90% probability if BLER is 10^{-6}, which is lower than that of Full Offloading scheme. Furthermore, the lowest SFP that PCP-based scheme can achieve is higher than other three schemes. It means that, when the targeted SFP is relatively low, PCP-based scheme can hardly provide feasible solution. For instance, it cannot fulfill the targeted SFP of 5×10^{-4} at all if BLER is 10^{-7} and $\alpha = 0.8$, while other three schemes can fulfill SFP below 5×10^{-4} with the probability of more than 95%.

Assume the targeted BLER is 10^{-7} and the latency constraint is $\alpha = 0.6$, the impact of the number of ENs on SFP performance in the face recognition application is presented in Fig. 13. The reliability of schemes is improved with the number of ENs increasing. When $N = 3$, the probability of Heuristic algorithm fulfilling 4.5×10^{-4} SFP is around 90%. The reliability of Heuristic algorithm scheme is much better

than that of Full Offloading which achieves 4.5×10^{-4} SFP with 30% probability if $N = 3$. The PCP-based scheme cannot provide feasible solution at all to the problem when $\alpha = 0.6$.

VII. CONCLUSION

In this paper, we systematically model the reliability and latency of the code-partitioning offloading for MEC-enabled AR service, and design the IPSO-based algorithm and Heuristic algorithm to minimize the service failure probability of computation offloading subject to the latency constraint. The numerical results show that the Heuristic algorithm achieves performance close to that of the IPSO-based algorithm with much lower computational complexity. Compared with the state-of-the-art algorithms, the Heuristic algorithm significantly improves the probability to fulfill the targeted service failure probability in various network conditions. Furthermore, the proposed algorithm is applicable to diverse uRLLC use cases in uRLLC besides AR service, since the computation tasks in many use cases can be modeled by directed acyclic graphs. Future research direction includes how to efficiently allocate communication and computation resources in the MEC-based network for multiple MDs with heterogeneous services and requirements. The non-orthogonal multiple access (NOMA) assisted MEC paradigm is a promising method to further shorten the access latency and relieve the resource contention among users.

REFERENCES

[1] D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui, "Mobile augmented reality survey: From where we are to where we go," *IEEE Access*, vol. 5, pp. 6917–6950, 2017.

[2] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler, "Toward low-latency and ultra-reliable virtual reality," *IEEE Netw.*, vol. 32, no. 2, pp. 78–84, Mar./Apr. 2018.

[3] H. Flores *et al.*, "Mobile code offloading: From concept to practice and beyond," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 80–88, Mar. 2015.

[4] M. Billinghurst, A. Clark, and G. Lee, "A survey of augmented reality," *Found. Trends Hum.–Comput. Interact.*, vol. 8, nos. 2–3, pp. 73–272, 2015.

[5] W. Zhang and Y. Wen, "Cloud-assisted collaborative execution for mobile applications with general task topology," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 6815–6821.

[6] M. Deng, H. Tian, and B. Fan, "Fine-granularity based application offloading policy in cloud-enhanced small cell networks," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC)*, May 2016, pp. 638–643.

[7] S. E. Mahmoodi, R. Uma, and K. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Trans. Cloud Comput.*, pp. 1–13, 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7463066, doi: 10.1109/TCC.2016.2560808.

[8] 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.

[9] ETSI, "Mobile-edge computing: Introductory technical white paper," ETSI, Sophia Antipolis, France, White Paper, 2014, no. 1.

[10] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 1, no. 2, pp. 89–103, Jun. 2015.

[11] 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.

[12] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.

[13] O. Munoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading," *IEEE Trans. Veh. Technol.*, vol. 64, no. 10, pp. 4738–4755, Oct. 2015.

[14] T.-C. Chiu, W.-H. Chung, A.-C. Pang, Y.-J. Yu, and P.-H. Yen, "Ultra-low latency service provision in 5G fog-radio access networks," in *Proc. IEEE 27th Annu. Int. Symp. Pers., Indoor, Mobile Radio Commun. (PIMRC)*, Sep. 2016, pp. 1–6.

[15] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Aug. 2017.

[16] Y. Sun, Z. Chen, M. Tao, and H. Liu. (2018). "Communication, computing and caching for mobile VR delivery: Modeling and trade-off." [Online]. Available: https://arxiv.org/abs/1804.10335

[17] X. Yang *et al.*, "Communication-constrained mobile edge computing systems for wireless virtual reality: Scheduling and tradeoff," *IEEE Access*, vol. 6, pp. 16665–16677, 2018.

[18] A. Al-Shuwaili and O. Simeone, "Energy-efficient resource allocation for mobile edge computing-based augmented reality applications," *IEEE Wireless Commun. Lett.*, vol. 6, no. 3, pp. 398–401, Jun. 2017.

[19] Q. Liu, S. Huang, J. Opadere, and T. Han, "An edge network orchestrator for mobile augmented reality," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 756–764.

[20] J. Liu and Q. Zhang, "Edge computing enabled mobile augmented reality with imperfect channel knowledge," *IEEE Netw. Lett.*, to be published.

[21] S. M. Azimi, O. Simeone, O. Sahin, and P. Popovski, "Ultra-reliable cloud mobile computing with service composition and superposition coding," in *Proc. Annu. Conf. IEEE Inf. Sci. Syst. (CISS)*, Mar. 2016, pp. 442–447.

[22] C.-F. Liu, M. Bennis, and H. V. Poor, "Latency and reliability-aware task offloading and resource allocation for mobile edge computing," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2017, pp. 1–7.

[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] Y. Zhang, H. Liu, L. Jiao, and X. Fu, "To offload or not to offload: An efficient code partition algorithm for mobile cloud computing," in *Proc. IEEE 1st Int. Conf. Cloud Netw. (CLOUDNET)*, Nov. 2012, pp. 80–86.

[25] J. G. Andrews, A. Ghosh, and R. Muhamed, *Fundamentals of WiMAX: Understanding Broadband Wireless Networking*. London, U.K.: Pearson Education, 2007.

[26] X. Qiu, Y. Dai, Y. Xiang, and L. Xing, "A hierarchical correlation model for evaluating reliability, performance, and power consumption of a cloud service," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 46, no. 3, pp. 401–412, Mar. 2016.

[27] Y. Valle, G. K. Venayagamoorthy, S. Mohagheghi, J. C. Hernandez, and R. G. Harley, "Particle swarm optimization: Basic concepts, variants and applications in power systems," *IEEE Trans. Evol. Comput.*, vol. 12, no. 2, pp. 171–195, Apr. 2008.

[28] H. Wu, W. Knottenbelt, K. Wolter, and Y. Sun, "An optimal offloading partitioning algorithm in mobile cloud computing," in *Proc. Int. Conf. Quant. Eval. Syst.*, 2016, pp. 311–328.

[29] *Evolved Universal Terrestrial Radio Access (E-UTRA) for FDD Repeater Radio Transmission and Reception (Release 8)*, Standard 3GPP TS 36.106, 2009.

[30] M. Mezzavilla, M. Miozzo, M. Rossi, N. Baldo, and M. Zorzi, "A lightweight and accurate link abstraction model for the simulation of LTE networks in ns-3," in *Proc. 15th ACM Int. Conf. Modeling, Anal. Simulation Wireless Mobile Syst.*, 2012, pp. 55–60.

[31] T. Tobita and H. Kasahara, "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms," *J. Scheduling*, vol. 5, no. 5, pp. 379–394, 2002.

[32] L. Yang, J. Cao, S. Tang, T. Li, and A. T. S. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, Jun. 2012, pp. 794–802.

[33] P. Zhao, H. Tian, and B. Fan, "Partial critical path based greedy offloading in small cell cloud," in *Proc. IEEE 84th Veh. Technol. Conf. (VTC-Fall)*, Sep. 2016, pp. 1–5.

[34] *Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Layer Procedures (Release 13)*, Standard 3GPP TS 36.213, 2016.

JIANHUI LIU received the B.E. degree in electronic information engineering from Beihang University, Beijing, China, in 2012, and the M.E. degree from the National Digital Switching System Engineering and Technology Research and Development Center, China, in 2015. He is currently pursuing the Ph.D. degree with the Department of Engineering, Aarhus University, Denmark. His current research interests are ultra-reliable and low-latency communications, and mobile edge computing.

QI ZHANG received the M.Sc. and Ph.D. degrees in telecommunications from the Technical University of Denmark, Denmark, in 2005 and 2008, respectively. She is currently an Associate Professor with the Department of Engineering, Aarhus University, Denmark. Besides her academic experiences, she has various industrial experiences. Her research interests include tactile Internet, the IoT, URLLC, mobile edge computing, massive machine type communication, non-orthogonal multiple access, and compressed sensing. She was a Co-Chair of the Co-Operative and Cognitive Mobile Networks Workshop in the ICC Conference, from 2010 to 2015, and was a TPC Co-Chair of BodyNets, in 2015. She is serving as an Editor for the *EURASIP Journal on Wireless Communications and Networking*.