Energy- and Cost-Aware Offloading of Dependent Tasks With Edge–Cloud Collaboration for Human Digital Twin

Qiang Zhang, Yuye Yang, Changyan Yi, *Member, IEEE*, Samuel D. Okegbile, *Member, IEEE*, and Jun Cai, *Senior Member, IEEE*

***Abstract*—Due to the potential of revolutionizing a variety of human-centric services, human digital twin (HDT) is envisioned to become an important part of our daily life. The HDT applications need to frequently collect and process data obtained from individuals and their environment, analyzing each physical twin while updating its corresponding virtual twin, which will consume a large amount of computing, storage and sensing resources cumulatively. Meanwhile, running HDT applications, such as emotion recognition, naturally contains the executions of several dependent tasks. Considering the resource limitations of mobile terminals, we enable dependent task offloading to mitigate terminal load and reduce the latency of HDT applications. Specifically, this article proposes an energy- and cost-aware offloading algorithm for dependent tasks with edge–cloud collaboration to empower HDT applications. We show that the problem of dependent task offloading under constraints of service cost and terminal energy consumption is NP-hard. The complexity of task interdependency makes the offloading decision under dual constraints even more challenging. The proposed offloading algorithm first generates task paths based on task interdependency and computation load, deriving the initial solution. Then, task reassignment and CPU frequency scaling methods are utilized to further optimize the obtained solution. Simulation results illustrate that our approach can achieve better performance in terms of makespan and service success ratio compared to the existing approaches.**

***Index Terms*—CPU frequency scaling, dependent task offloading, edge–cloud collaboration, human digital twin (HDT).**

I. Introduction

VARIOUS mobile applications and smart mobile terminals have brought great changes to human production and life in recent years with the technical support of 5G, cloud computing, edge computing, sensor, smart operating system, and so on. At the same time, human digital twin (HDT), an emerging technique, is developing rapidly, and allows an in-silico representation of any individual with the ability to reflect molecular status, physiological status, emotional and psychological status, as well as lifestyle evolutions [1]. In HDT, a digital representation of a human being is implemented as a virtual twin (VT) in the virtual space [2]. The mapping and connection between such a human being and the associated VT are important for the interactions between human-VT pairs [3]. HDT will greatly facilitate human-centric systems while further enriching applications, such as virtual reality (VR), multimedia joint identification retrieval, mobile social games, and health monitoring using body sensor networks [4]. Due to the continual change of the body state of human beings, correspondingly the considerably long operation time of HDT applications in his or her mobile terminal is inevitable. Besides, these applications usually require intensive and continuous processing of massive data. Therefore, the rich functions of HDT run on mobile terminals, e.g., smart phones, smart watches and smart helmets, may also cause higher battery usage and longer processing delays. As a result, the terminal energy saving of HDT applications is critical given the limited power constraints of these terminal devices.

It is worth noting that HDT applications require the execution of multiple dependent tasks. A typical example is the emotion recognition (ER). ER can analyze a video of a person to identify his or her current emotion. ER contains multiple dependent tasks in the process of performing feature extraction, machine learning (ML), and so on [5]. The dependency among tasks enforces that each task must be executed after the execution of all its predecessor tasks has been finished [6]. Besides, each task needs the output of all its predecessor tasks as the input to process. Moreover, these tasks commonly have different computation loads and output data sizes. Therefore, it is difficult to execute all tasks in parallel completely. In addition, the computing capability of a mobile terminal is difficult to perform tasks with heavy loads although improving the capabilities of such mobile terminals continues to receive attention. To tackle the limitations related to hardware conditions of mobile terminals and speed up the running of latency-sensitive applications, task offloading has been extensively studied in [7]. However, in our considered

Manuscript received 7 February 2024; revised 26 April 2024 and 18 May 2024; accepted 24 May 2024. Date of publication 28 May 2024; date of current version 23 August 2024. This work was supported in part by the State Key Laboratory of Massive Personalized Customization System and Technology under Grant H&C-MPC-2023-04-01; in part by the Postgraduate Research and Practice Innovation Program of NUAA under Grant xcxjh20231601; in part by the National Natural Science Foundation of China under Grant 61802181; and in part by the Natural Science Foundation of Jiangsu Province of China under Grant BK20222012 and Grant BK20231439. *(Corresponding author: Changyan Yi.)*

Qiang Zhang, Yuye Yang, and Changyan Yi are with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China (e-mail: cszhangqiang@126.com; mryyy@nuaa.edu.cn; changyan.yi@nuaa.edu.cn).

Samuel D. Okegbile and Jun Cai are with the Department of Electrical and Computer Engineering, Concordia University, Montreal QC H3G 1M8, Canada (e-mail: samuel.okegbile@concordia.ca; jun.caig@concordia.ca).

Digital Object Identifier 10.1109/JIOT.2024.3406591

problem, due to the dependency among tasks and the large amount of potential solutions which increases in exponential order with the number of tasks, the optimal offloading decision cannot be derived in polynomial time. Meanwhile, the data transmission when offloading from the mobile terminal will consume some energy, while the constraint of terminal energy consumption for HDT further raises the difficulty of the problem.

Compared with tasks offloading to a single remote cloud server, offloading to edge servers can result in reduced service access delays attributed to the local area networks. However, it is important to note that the computational resources available in edge servers are significantly less than those in a cloud server. As a solution to this limitation, the concept of edge–cloud collaboration has been introduced to enhance the elasticity of services for both individual users and public systems [8]. When considering the prolonged execution time of HDT applications involving numerous users, the service cost of an edge–cloud collaboration system can escalate, leading to an increase in carbon emissions. Furthermore, if the service costs are not carefully managed, the resources within any edge server may be quickly depleted, resulting in the inability to accommodate new requests due to a lack of available resources. Hence, it is imperative not to overlook the aspect of service cost during the process of offloading dependent tasks for HDT applications.

In this article, we aim to solve the problem of dependent task offloading for HDT. Since the status of a physical twin often changes, such as emotions, the VT also needs to update its status correspondingly. These characteristics indicate that the running time of HDT is generally longer, which can consume a large amount of terminal energy. Besides, if edge–cloud collaboration is used, the service cost cannot be ignored. Consequently, for a HDT application with edge–cloud collaboration, both terminal energy consumption and service cost need to be considered. Meanwhile, this article focuses on dependent task offloading due to the complex and evolutionary functions of HDT rather than independent tasks. To address this, we propose a novel scheme called energy- and cost-aware offloading (ECAO) scheme, specifically designed for dependent tasks. ECAO incorporates the dynamic voltage and frequency scaling (DVFS) technique to enhance the optimization of makespan and terminal energy consumption. The main contributions of this article are summarized in the following: 1) the problem of dependent task offloading for HDT is formulated as an online optimization problem to minimize makespan under dual constraints of terminal energy consumption and service cost, where the NP-hardness of the problem is also proved; 2) to tackle the complex task dependency, searching task paths with special dependency is designed to generate an initial offloading solution; 3) iterative optimization of service cost and makespan is designed to implement a task rescheduling scheme to further improve the obtained solution; 4) two task rescheduling algorithms are designed to strike the balance between makespan and terminal energy consumption, where the latest and other output arrival times of specific predecessors are analyzed and utilized to combine with DVFS to adjust CPU frequency for suitable local tasks; and 5) extensive simulations are conducted to show that the proposed ECAO scheme can outperform counterparts in terms of makespan and service success ratio.

The remainder of this article is organized as follows. In Section II, the related work is summarized. Section III describes the system model and problem formulation. The ECAO scheme is proposed in Section IV. Section V elaborates performance evaluation. Finally, the conclusion is given in Section VI.

II. Related Work

HDT, as an innovative technology, holds the potential to significantly enhance both production and the quality of life for individuals. It encompasses various domains, including social networks, VR, intelligent manufacturing, and personalized healthcare [9]. HDT creates a VT for users, complete with simulated body organs and habits within the virtual space. By leveraging smart terminals, HDT can sense and provide feedback on human states and environmental conditions [10]. To enable intelligent services, ML algorithms, such as federated learning [11], are integrated into HDT. However, these ML algorithms may impose substantial computational demands and introduce high latency, failing to meet the requirements of HDT. As shown in Fig. 1, ER contains multiple dependent tasks, such as preprocessing, framing, time domain extraction, frequency domain extraction, grayscale, histogram, feature extraction, clustering, fusion, and so on. An offloading decision-making scheme is responsible for determining the execution node for each task so as to minimize the makespan under constraints of terminal energy consumption and service cost. For better clarity, the related work is categorized into two: 1) dependent task offloading and 2) independent task offloading.

A. Dependent Task Offloading Schemes

Guo et al. [12] proposed a multi-UAV cooperative communication and computing optimization (MCCCO) scheme. MCCCO decomposed the original problem into two subproblems, i.e., computation offloading and resource allocation. The authors utilized a two-layer game-theoretic approximation offloading and convex optimization tools to solve the subproblems. This work aimed to minimize makespan under UAV energy consumption constraint. However, the service cost was not studied. Besides, this work did not prove that MCCCO can reach a Nash equilibrium within a Polynomial time. Hence, MCCCO may expend a large amount of time in order to reach the convergence of MCCCO, which is not very suitable for a real-time system. Chen et al. [13] proposed a dependency-aware task offloading method based on deep Q-networks (DODQ) with edge–cloud collaboration. DODQ customized deep Q-networks to train the decision-making model of task offloading, and the authors considered the parallelism of tasks without presetting the task priority when scheduling tasks. However, this work only focused on the optimization of makespan and did not study terminal energy consumption and service cost, which may be not suitable for mobile terminals with limited battery capacity.

Moreover, the trained neural network parameters may be not efficient for a new execution environment if the difference between the new environment and the trained environment is larger. Feng et al. [14] proposed a dependency-aware task offloading algorithm with task reconfiguration (PRAISE) algorithm to maximize the total offloading benefit. PRAISE transformed a task graph into multiple sequential layers, and utilized convex optimization to allocate resources in terms of computation, communication and time slots. However, the candidate offloading position was selected first according to the distances among physical nodes. For a subtask, if the data size is relatively small and the computation load is larger, the above method may missed better solutions if a distant edge server has a more powerful computing capability. In addition, terminal energy consumption was not considered, which can result in the insufficiency of terminal battery power. Wang et al. [15] proposed a heuristic computation offloading scheme for dependent tasks to minimize the system latency with energy consumption consideration in multiaccess edge computing. The proposed three-stage offloading scheduling algorithm contained serialization subtask stage, priority calculation stage and server selection stage. However, a single task-greedy scheduling was adopted to implement local latency minimization in the server selection stage, which probably missed some better solutions. Besides, the service cost was not considered, which leads to the edge servers may not afford the service due to the limited resources. Yan et al. [16] introduced a deep reinforcement learning framework for joint optimization of offloading decision and resource allocation. In the proposed framework, a Gaussian noise-added order-preserving quantization method was utilized to generate offloading actions. In this work, the objective was to minimize the weighted sum of the terminal energy consumption and task execution time, which was a multiobjective optimization problem. But, the authors did not consider the constraint of service cost. Liu et al. [17] proposed an online offloading framework with an event-driven scheduling mechanism for multiple mobile applications with edge–cloud collaboration, utilizing a heuristic ranking-based algorithm to minimize the average makespan of offloaded applications. The proposed scheme achieved less makespan compared to the existing methods, but this work did not study terminal energy consumption and service cost. Mo et al. [18] augmented a deep reinforcement learning model with graph convolutional networks to optimize the makespan of applications by describing task dependencies. Zhao et al. [19] proposed a convex programming-based algorithm for offloading dependent tasks with service caching and introduced a favorite successor-based algorithm for a special case with homogeneous MEC. However, the service cost and terminal energy consumption were not explored. Nguyen et al. [20] investigated dependent task offloading and communication resource allocation to minimize user average latency, employing a metaheuristic method for offloading and convex optimization for resource allocation. Nevertheless, the energy consumption of users' devices and service cost were not considered. Lv et al. [21] proposed a table-based task offloading algorithm for the dependent task offloading problem under energy consumption and service caching constraints in mobile edge computing. Tang et al. [22] formulated the task offloading decision as a Markov decision process, and proposed a task priority and deep reinforcement learning-based task offloading algorithm with edge–cloud collaboration. Unfortunately, the service cost of edge servers was not considered in [21] and [22]. To summarize, all of the above works did not consider at least one of terminal energy consumption and service cost. Besides, most deep reinforcement learning approaches can not efficiently adapt to a new execution environment.

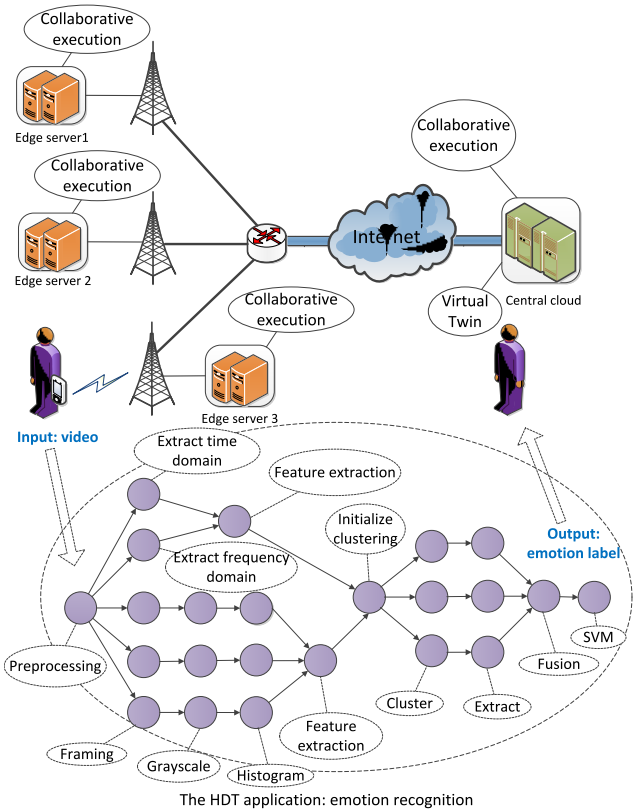

Fig. 1. System architecture of dependent task offloading for HDT applications.

B. Independent Task Offloading Schemes

In contrast to dependent task offloading, the researches on independent task offloading do not have the difficulty to tackle task dependency. Meanwhile, the ways to calculate the makespan of the two kind of offloading are different, and it is relatively easy to derive the makespan for an independent task offloading solution. The specific related work about independent task offloading is described as follows. Dinh et al. [23] introduced a computation offloading approach enabling a single mobile device to distribute independent tasks across multiple access points, aiming to minimize both the device's energy consumption and total latency. Tran and Pompili [24] addressed joint task offloading and resource allocation in mobile edge computing, presenting a heuristic algorithm for offloading decision on independent tasks. Li et al. [25] proposed computation replication to expedite result downloading in task offloading, but this approach increased edge node service costs, making it unsuitable for

certain applications. Malik and Vu [26] focused on energy optimization for computation offloading in massive MIMO-enabled edge networks, utilizing data partitioning, transmit power control and CPU frequency scaling. Ale et al. [27] introduced an end-to-end deep reinforcement learning model to maximize computational tasks before deadlines and minimize energy consumption. Chen et al. [28] presented a deep reinforcement learning method for joint optimization of computation offloading and resource allocation in mobile edge computing, incorporating a temporal attentional deterministic policy gradient. Yang et al. [29] proposed a multitask learning-based feedforward neural network for optimal computation offloading in the MEC system, formulating the offloading decision problem as a multiclass classification issue. Cao et al. [30] introduced a multiagent deep reinforcement learning scheme for multichannel access and task offloading in MEC-enabled industry 4.0, focusing on reducing computation delay and improving channel access success rate. Ren et al. [31] presented a deep reinforcement learning-based approach for offloading independent tasks to fog access points, with the objective of minimizing long-term system energy consumption. Wang et al. [32] proposed a multiagent generalized adversarial imitation learning-based computation offloading algorithm in pervasive edge computing networks, aiming to minimize the average task completion time for each device. To summarize, all of the above researches are not suitable for dependent task offloading since there is no dependency among tasks in these studies.

Based on the above analysis of the related work, the existing methods did not adequately study the service cost of dependent task offloading in edge–cloud collaboration system. Meanwhile, if a new execution environment appears, the running time of deep reinforcement learning methods may not satisfy the real time requirement of latency-sensitive applications. Beside, some heuristic algorithms employed single task-greedy-based scheduling to make offloading decision, which is also not efficient compared to multiple-task scheduling. Considering the characteristics of HDT and edge–cloud collaboration system, both terminal energy consumption and service cost can not be neglected. Since the task dependency and the above dual constraints are difficult to deal with, it is important to analyze task graph and generate effective offloading solution in real time. In this article, the proposed ECAO scheme focuses on the dual constraints of terminal energy consumption and service cost, and the objective is to minimize the makespan of dependent task offloading for HDT. ECAO searches task paths with special dependency to generate an initial offloading solution. Besides, ECAO adopts DVFS and iterative optimization with constraints to further improve the initial offloading solution. To summarize, ECAO generates its offloading solution based on special task paths, iterative optimization and CPU frequency scaling, which distinguishes it from the existing methods.

III. System Model and Problem Formulation

In this section, the system model and the problem formulation are presented. The system model consists of task model, energy consumption model and service cost model. The main notations used in this article are shown in Table I.

TABLE I
Notation Definition

Notations	Definition
L_i	The computation load of task i
$D_{i,j}$	The data size of edge (i, j)
$pred(i)$	The set of predecessors of task i
$succ(i)$	The set of successors of task i
ET	The set of exit tasks
$R_{x,y}$	The transmission rate from node x to node y
$TT(i,j)$	Transmission time of $D_{i,j}$
z_i^k	The offloading decision variable
T_k^r	The ready time of execution node k
T_i^s	The start time of task i
T_i^k	The execution time of task i on node k
T_i	The execution time of task i based on decision
S_k	The processing speed of execution node k
FT_i	The finish time of task i
P_a	The downlink transmission power of the access point
P_u	The uplink transmission power of the access point
G_d	The channel gain when downloading data
G_u	The channel gain when uploading data
$D_s(Z)$	The service delay corresponding to decision Z
E_c	The terminal energy consumption of the local execution
E_u	The terminal energy consumption of uploading data
E_d	The terminal energy consumption of downloading data
C_s	The total service cost
C_a	The constraint of service cost
E_a	The constraint of termianl energy consumption

A. System Model

Fig. 1 shows the system architecture. An HDT application is modeled as a directed acyclic graph (DAG), denoted by DAG=(V, E), where the vertex set $V = \{v_1, v_2, \ldots, v_N\}$ represents individual tasks and edge set E denotes the data dependencies among these tasks. The execution nodes, i.e., the mobile terminal, edge servers, central cloud are represented by $U = \{u_1, u_2, \ldots, u_M\}$, where u_1 denotes the mobile terminal. We define z_i^k as the offloading decision variable, which is a binary value. If task i is allocated to the execution node k, $z_i^k = 1$, and otherwise $z_i^k = 0$. Similar to [33], the execution time of task i can be obtained as

$$T_i = \sum_{k=1}^{M} z_i^k \frac{L_i}{S_k} \quad \forall v_i \in V \tag{1}$$

where L_i denotes the computation load of task i, while S_k is the processing speed of execution node k.

For the data $D_{i,j}$ corresponding to the edge (i, j) in the DAG, we denote x as the sender and y as the receiver. Then, the transmission time of this data can be expressed as

$$\mathrm{TT}(i, j, x, y) = \sum_{x=1}^{M} \sum_{y=1}^{M} z_i^x z_i^y \frac{D_{i,j}}{R_{x,y}}, x \neq y \tag{2}$$

where $R_{x,y}$ denotes the transmission rate from node x to node y. When the sender and the receiver are the same node, the transmission time is zero. We denote W as the fixed bandwidth of the orthogonal channels allocated to the mobile terminal [34]. P_a and P_u denote the downlink and uplink transmission power of the access point, respectively. Specifically, according to

Shannon theory [35], the downloading rate and uploading rate can be obtained by

$$R_{x,1} = W \log_2 \left(1 + \frac{P_a G_d}{\sigma^2}\right), x \neq 1 \tag{3}$$

$$R_{1,y} = W \log_2 \left(1 + \frac{P_u G_u}{\sigma^2}\right), y \neq 1 \tag{4}$$

where G_d and G_u are the channel gains when downloading and uploading data, respectively [36], while σ^2 denotes the white Gaussian noise power [37].

The finish time of a task is determined by its start time and execution time. Since a task may contain multiple predecessor tasks, its start time depends on the latest output arrival time of its predecessors and the ready instant of its execution node [38]. Therefore, the finish time of task j can be calculated as follows:

$$\mathrm{FT}_j = \max\left\{\sum_{k=1}^{M} z_j^k T_k^r, \max_{i \in \mathrm{pred}(j)} \{\mathrm{FT}_i + \mathrm{TT}(i,j)\}\right\} + T_j. \tag{5}$$

An application is completed only if all exit tasks are finished. So, the latest completion time of the exit tasks determines the service delay, i.e., makespan which can be given by

$$D_s(Z) = \max_{h \in ET}\left\{\max\left\{\sum_{q=1}^{M} z_h^q T_q^r, \max_{i \in \mathrm{pred}(h)} \{\mathrm{FT}_i + \mathrm{TT}(i,h)\}\right\} + \sum_{k=1}^{M} z_h^k \frac{L_h}{S_k}\right\}. \tag{6}$$

In addition, according to the research in [39], the terminal energy consumption of the local execution can be calculated as follows:

$$E^c = \sum_{i=1}^{N} z_i^1 \delta L_i S_1^{i^2} \tag{7}$$

where δ is the terminal CPU parameter.

The terminal energy consumption of uploading data can be expressed as

$$E^u = \sum_{(i,j) \in E} \sum_{y=2}^{M} z_i^1 z_j^y \frac{D_{i,j}}{R_{1,y}} p_u \tag{8}$$

where p_u is the uploading power of the terminal. Similarly, the terminal energy consumption of downloading data can be expressed as

$$E^d = \sum_{(i,j) \in E} \sum_{x=2}^{M} z_i^x z_j^1 \frac{D_{i,j}}{R_{x,1}} p_d \tag{9}$$

where p_d is the downloading power of the terminal.

To describe the usage of computational resource in the edge–cloud collaboration system, we utilize the price mechanism of cloud services to define service cost [40], which is given by the following equation:

$$C_s = \sum_{i=1}^{N} \sum_{k=2}^{M} z_i^k \frac{L_i}{S_k} c_k \tag{10}$$

where c_k is the cost of computational resource usage of node k per time unit. Considering that service cost is utilized to quantify the usage of computational resource in edge servers and the cloud server, so the service cost does not involve the cost of computational resource usage in the mobile terminal, i.e., k is greater than 1 in (10).

B. Problem Formulation

The problem of dependent task offloading targeted at minimizing the service delay, while considering constraints related to service cost and terminal energy consumption for HDT can be formulated as

$$\min \ D_s(Z) \ \text{s.t.} \tag{11}$$

$$\sum_{i=1}^{N} z_i^1 \delta L_i S_1^{i^2} + \sum_{(i,j) \in E} \sum_{y=2}^{M} z_i^1 z_j^y \frac{D_{i,j}}{R_{1,y}} p_u + \sum_{(i,j) \in E} \sum_{x=2}^{M} z_i^x z_j^1 \frac{D_{i,j}}{R_{x,1}} p_d \leq E_a \tag{11a}$$

$$\sum_{i=1}^{N} \sum_{k=2}^{M} z_i^k \frac{L_i}{S_k} c_k \leq C_a \tag{11b}$$

$$\max_{i \in \mathrm{pred}(h)} \{\mathrm{FT}_i + \mathrm{TT}(i,h)\} \leq T_h^s \quad \forall v_h \in V \tag{11c}$$

$$\sum_{k=1}^{M} z_i^k = 1 \quad \forall v_i \in V \tag{11d}$$

$$z_i^k \in \{0,1\} \quad \forall v_i \in V \ \forall u_k \in U. \tag{11e}$$

Here, (11a) is the constraint of terminal energy consumption, and it specifies that the sum of terminal energy consumption, including uploading, downloading, and local execution, is bounded by the required energy consumption E_a. Constraint (11b) denotes that the total service cost should be equal to or less than the required service cost C_a. Constraint (11c) is the dependency constraint, which indicates that a task can not start if all predecessors have not been finished. Constraint (11d) is the assignment constraint, and it denotes that each task can be assigned to only one execution node. Constraint (11e) specifies that the decision variable is a binary variable. For convenience, we define the objective (11) and the corresponding constraints as problem P1. In the following, the multiprocessor scheduling problem is first defined, and then based on this definition, we show that problem P1 is NP-hard.

Definition 1 [Multiprocessor Scheduling Problem (MPSP)]: Given N tasks and M processors, each task i on processor j has an execution time t_{ij}, consumes an energy e_{ij} and expends a service cost c_{ij}. The objective of MPSP is to minimize the makespan of the input tasks subject to the constraints of energy consumption and service cost. It can be formulated as

$$\min \max\{\sum_{j=1}^{M}\left(t_j^r + t_{ij}\right)x_{ij}\} \quad \forall i \in \{1,2,\ldots,N\} \tag{12}$$

$$\text{s.t.} \ \sum_{i=1}^{N} e_{i1} x_{i1} \leq E \tag{12a}$$

$$\sum_{i=1}^{N}\sum_{j=2}^{M} c_{ij}x_{ij} \le C \tag{12b}$$

$$\sum_{j=1}^{M} x_{ij} = 1 \quad \forall i \in \{1, 2, \ldots, N\} \tag{12c}$$

$$x_{ij} \in \{0, 1\}\ \forall i \in \{1, 2, \ldots, N\} \quad \forall j \in \{1, 2, \ldots, M\} \tag{12d}$$

where t_j^r denotes the instant when processor j becomes ready and x_{ij} is the decision variable. If task i is allocated to processor j, $x_{ij} = 1$, and otherwise $x_{ij} = 0$. Constraint (12a) is the energy consumption constraint of the designated processor 1, and (12b) denotes the total service cost constraint of candidate processors except processor 1. Constraint (12c) means that each task only can be allocated to one processor. Constraint (12d) is the binary constraint.

Theorem 1: Problem P1 is NP-hard.

Proof: We consider a special case of P1, which remove all task dependency from a dependent task graph. Hence, the above special case ensures that tasks can be executed independently. The special case of problem P1 can be defined as P2, which does not have (11c). The tasks of P2 correspond to the input tasks of MPSP. The execution nodes of P2 correspond to the processors of MPSP. Besides, the mobile terminal of P2 corresponds to the designated processor of MPSP, and the cloud server and edge servers correspond to the other candidate processors. The energy consumption constraint of P2 is equivalent to the first constraint of MPSP. The service cost constraint of P2 is equivalent to the second constraint of MPSP. The assignment and binary constraints of P2 and MPSP are the same. Minimizing the service delay of P2 is equivalent to the objective of MPSP. Since the MPSP is NP-hard [41], problem P2 is also NP-hard. Considering P2 is a special case of P1, the original problem P1 is also NP-hard. ■

Remark 1: It is infeasible to find the optimal solution to problem P1 within polynomial time due to the NP-hardness of P1. In order to obtain a near-optimal solution in a real-time manner, in the following section, we propose an ECAO scheme based on some heuristic methods. The main idea of ECAO is to utilize divide and conquer method to iteratively generate local decision for special task paths. Besides, task rescheduling based on performance tradeoff is designed to overcome the local optimality. Moreover, ECAO adopts DVFS to further enhance the performance.

IV. Proposed ECAO Scheme

The proposed ECAO scheme aims to devise a task-offloading solution that minimizes application completion time while adhering to the dual constraints of energy consumption and service cost. Initially, the priority of the target task is calculated, and multiple special task paths are generated. For the offloading decision on each task path, the execution site is determined based on minimal completion time. Subsequently, the adherence of this decision to the service cost constraint is assessed. If the criterion is not met, ECAO will improve the current solution by task rescheduling. If the service cost constraint is satisfied, ECAO will further optimize application completion time by striking a balance among makespan, service cost, and energy consumption. This optimization involves the utilization of task reassignment and CPU frequency scaling methods. The flowchart illustrating the ECAO scheme is depicted in Fig. 2.

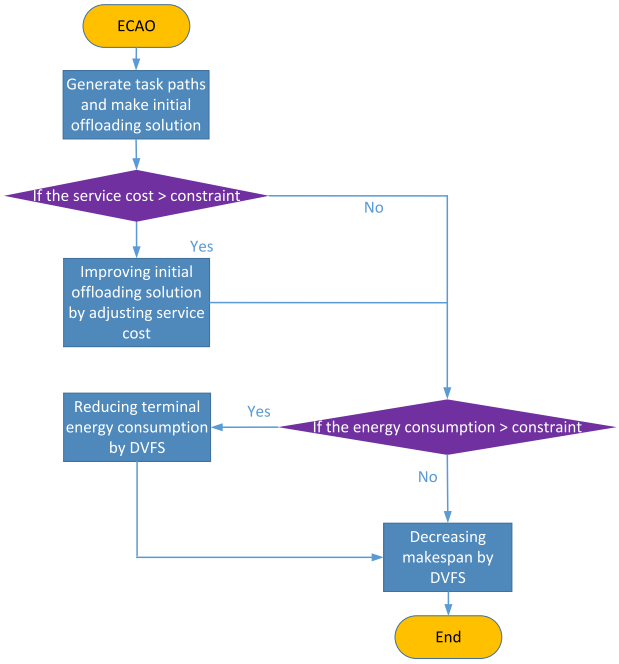

Fig. 2. Flowchart of ECAO.

A. Target Task Sorting

First, all tasks need to be sorted according to their variance in execution time. Given that all tasks have their predecessors executed locally, the execution time of each task is computed on individual worker nodes using (1). Next, the average execution time difference between v_i on each cloud node and on the local terminal is calculated, and the tasks are stored in descending order of execution time difference. The execution time difference is calculated as follows:

$$P_i^1 = \sum_{k=2}^{M} \frac{\left|T_i^k - T_i^1\right|}{M-1}. \tag{13}$$

B. Task Path Generation and Local Scheduling

The target task with the highest priority is selected as the destination task of the task paths. The entry task is selected as the source task. If there are multiple entry tasks, a virtual task is added into the task graph as the new entry task. ECAO searches paths from the source task to the destination task by depth first search (DFS) algorithm for H times, and the size of H is controlled reasonably to guarantee the acceptable complexity. Then, several task paths are generated. The task path with higher computation load is refined first based on task dependency. When a qualified task path is generated, a local scheduling is made to minimize the makespan of the task path. ECAO searches task paths and makes local scheduling iteratively till all tasks are scheduled. The details of the above method is demonstrated in Algorithm 1.

Algorithm 1 Dependent Task Offloading Based on Special Task Paths

Input: DAG, execution nodes, *List*, and I_s.
Onput: Initial offloading decision Z.

```
Sort target tasks and add them into List;
for each v_i in List
    Search H paths at most from source to v_i by DFS;
    Sort the above paths in descending order of workload;
    for each path j in sorted paths
        for each v_k on path j
            if v_k is not in set I_s
                if ∃v_q ∈ pred(k) outside path j is not in I_s
                    Break;
                end if
                Mark v_k as a valid task on path j;
            end if
        end for
        Make local scheduling;
        Add valid tasks on path j into I_s;
    end for
end for
```

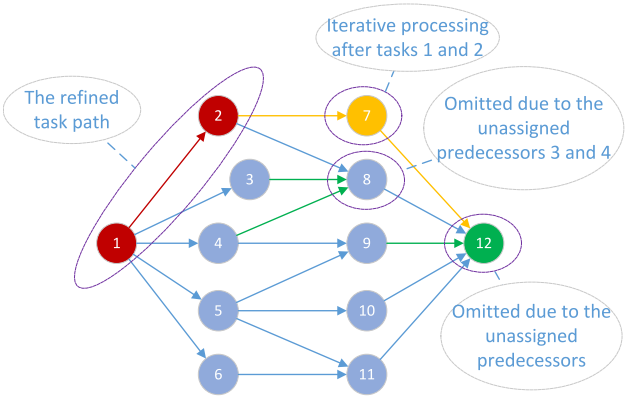

Fig. 3. Key steps of dependent task offloading based on special task paths.

The key steps of Algorithm 1 are described as follows.

1) *Sort Target Tasks:* As shown in Fig. 3, all tasks are sorted by Algorithm 1. We assume that the result is (12, 2, 8, 1, 7, 4, 10, 5, 6, 3, 9, 11). Then, the first target task, i.e., task 12 is selected as the destination task, and task 1 is selected as the source task.
2) *Search and Sort Task Paths:* Algorithm 1 searches six paths from task 1 to task 12, and they are (1, 2, 7, 12), (1, 2, 8, 12), (1, 3, 8, 12), (1, 4, 8, 12), (1, 4, 9, 12), and (1, 5, 9, 12). Then, the task path with the maximal computation load is assigned with the highest priority. We assume that the path (1, 2, 8, 12) has the highest priority, and the following path is (1, 2, 7, 12).
3) *Refine the Current Task Path:* Tasks on the path (1, 2, 8, 12) is judged sequentially to refine the task path. I_s denotes the task set that has been scheduled, and it is initialized as an empty set. As shown in Fig. 3, task 1 and task 2 satisfy the requirement of the special path. Task 8 has a predecessor task 3 which is not in I_s, so task 8 is not retained on the current path. Similarly, task 12 has a predecessor task 9 which has not been scheduled, therefore task 12 is not retained. Finally, the refined task path is (1, 2).
4) *Make Local Scheduling:* Algorithm 1 makes local scheduling on the task path (1, 2) to derive its local minimal makespan. Specifically, task 1 and task 2 as a whole are allocated to each execution node, and the execution node with the minimal makespan is selected. Then, task 1 and task 2 are added into I_s.
5) *Iterative Processing:* After dealing with the task path (1, 2), Algorithm 1 continues to process the task path (1, 2, 7, 12). As shown in Fig. 3, because task 1 and task 2 are in I_s, they are not processed further. Considering that task 12 has a predecessor task 9 which has not been scheduled, the current task path only retains task 7. Local scheduling is made on task 7 so that the execution node with the minimal makespan is selected. Algorithm 1 processes task path iteratively till all tasks have been added into I_s.

C. Improving Initial Offloading Solution by Adjusting Service Cost

This section outlines a task rescheduling scheme aimed at enhancing the initial offloading solution by adjusting service costs. The proposed rescheduling scheme comprises two primary functions. The first function focuses on generating a feasible solution in terms of service cost if the initial offloading solution proves unfeasible. The second function aims to further reduce the makespan based on the updated offloading solution.

Upon obtaining the initial offloading solution, the ECAO scheme evaluates the corresponding makespan and service cost. If the service cost exceeds the predefined constraint, ECAO triggers the first function. During the first function, tasks are sorted in descending order of service cost, and each task is rescheduled accordingly. A cost-greedy policy is employed, selecting the execution node with the lowest service cost to replace the original execution node. This policy iteratively continues until all tasks are rescheduled.

If the service cost is equal to or less than the constraint, ECAO initiates the second function. In this phase, tasks are sorted in descending order of execution time, and each task is rescheduled accordingly. The rescheduling scheme selects the execution node with the minimal makespan from the feasible node set for each task iteratively. The feasible node set includes execution nodes capable of providing services while adhering to the service cost constraint. The algorithm's specifics are presented in Algorithm 2.

The key steps of Algorithm 2 are outlined as follows: if the service cost exceeds the constraint value, tasks are sorted in descending order of service cost, resulting in the scheduling queue (12, 8, 7, 4, 10, 5, 6, 3, 9, 11, 1, 2). Task 12 is assigned to the execution node with the least service cost. Subsequently, Algorithm 2 assesses whether the updated service cost meets the constraint. If the constraints are met, tasks are sorted in descending order of execution time, and the sorted tasks are added to the rescheduling queue. The execution node with the minimal makespan is selected iteratively for each task in the rescheduling queue, ensuring it satisfies the service cost

Algorithm 2 Improving Initial Offloading Solution by Task Rescheduilng

Input: Initial offloading solution, service cost constraint.
Onput: Improved offloading decision.

if the current service cost > constraint
  Sort tasks in descending order of service cost;
  Establish the scheduling queue;
  for each v_i in the scheduling queue
    Perform cost-greedy policy;
    Update the current service cost;
    if the current service cost ≤ constraint
      Break;
    end if
  end for
  Sort tasks in descending order of execution time;
  Establish the rescheduling queue;
  for each v_i in the rescheduling queue
    Perform feasible makespan-greedy policy;
  end for
end if

Algorithm 3 Reducing Terminal Energy Consumption by DVFS

Input: Current offloading solution, energy constraint.
Onput: Improved offloading decision, CPU frequency for terminal tasks.

if the current energy consumption > constraint
  for each v_i in the terminal processing queue S_q
    for each v_j in succ(i)
      if $\mathrm{FT}_i + \mathrm{TT}(i,j) = T_j^s$
        v_i is unsuitable and added to S_u;
        Break;
      end if
    end for
  end for
  Generate suitable task set $S_s = S_q - S_u$
  Calculate priority for each task in S_s by Eq. (14);
  Establish the priority queue;
  for each v_i in the priority queue
    Calculate the CPU frequency S_0^i by Eq. (17);
  end for
end if

constraint. Otherwise, Algorithm 2 continues with the cost-greedy policy on the next task in the scheduling queue.

D. Reducing Terminal Energy Consumption by DVFS

After completing Algorithms 1 and 2, there is a possibility that the terminal energy consumption exceeds the specified constraint. To generate a feasible solution which satisfies dual constraints, we propose an algorithm aimed at reducing the CPU frequency for suitable local tasks, thereby mitigating terminal energy consumption. According to the inherent characteristics of task dependency, if a task v_k has multiple predecessors, the latest output arrival time of predecessors will affect the start time of v_k. But, if a predecessor v_i of v_k does not have the latest output arrival time compared to the other predecessors, the suitable extension of its execution time will not extend the finish time of v_k. Considering that the above predecessor v_i may be also a predecessor of another task, so all successors of v_i need to be analyzed to determine the adjustable execution time of v_i. The primary procedure is outlined below. Initially, ECAO identifies tasks executed on the mobile terminal that are suitable for a further reduction in CPU frequency. Subsequently, ECAO sorts these identified tasks to establish a priority for scheduling. Finally, ECAO adjusts the CPU frequency for each task in the established order, considering its specific time constraint without extending the original makespan.

In the initial phase, ECAO computes the time difference to assess which tasks are appropriate for adjustment. For a task v_i executed on the mobile terminal, it is deemed suitable if the start time of all successor tasks of v_i occurs later than the output arrival time of v_i. Conversely, if any start time precedes the output arrival time, adjusting the CPU frequency for v_i is deemed unsuitable.

In the subsequent step, the scheduling priority for task v_i is determined by

$$P_i^2 = \frac{E_i^p - E_i^q}{T_i^q - T_i^p} \tag{14}$$

where E_i^p and E_i^q denote the energy consumption of terminal execution with the original and minimal CPU frequency for v_i, respectively, and T_i^p and T_i^q are correspondingly the task finish time of v_i with the original and minimal CPU frequency.

In the next step, ECAO analyzes the specific time constraint for each suitable task to obtain the corresponding CPU frequency. The minimum start time of succ(i) is represented as $\min\{T_j^s\}$, where $v_j \in \mathrm{succ}(i)$. To guarantee no increase of the makespan, two conditions need to be considered when calculating the time constraint. The first condition is that the updated output arrival time of task i should be earlier or equal to the minimum start time of succ(i). The second condition is the execution of task i should not prolong the following tasks in the processing queue of the mobile terminal. Based on the above analysis, the specific time constraint FT_i' for v_i is defined by

$$\mathrm{FT}_i' + \mathrm{TT}(i,r) \le \min\{\min\{T_j^s\}, T_k^s\} \tag{15}$$

where task r has the minimum of the start time of succ(i) and task k is the next task of task i in the processing queue of the mobile terminal. We rewrite (15) by substituting the updated CPU frequency S_1^i as

$$\mathrm{FT}_i - \frac{L_i}{S_1} + \frac{L_i}{S_1^i} + \mathrm{TT}(i,r) \le \min\{\min\{T_j^s\}, T_k^s\} \tag{16}$$

where S_1 denotes the normal CPU frequency of the mobile terminal and FT_i is the original task finish time. When (16) satisfies the equality condition, the efficiency of CPU frequency scaling is maximized. Therefore, the updated CPU frequency for v_i can be calculated as

$$S_1^i = \frac{L_i}{\frac{L_i}{S_1} + \min\{\min\{T_j^s\}, T_k^s\} - \mathrm{TT}(i,r) - \mathrm{FT}_i}. \tag{17}$$

The details of the above method is demonstrated in Algorithm 3.

The key steps of Algorithm 3 are outlined below. Fig. 4 illustrates the scenario where task 5 resides in the terminal processing queue. The set succ(5) contains tasks 9, 10, and

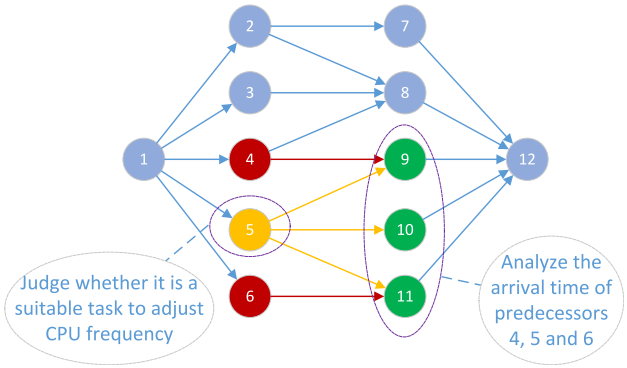

Fig. 4. Adjusting CPU frequency for suitable tasks.

11, which are subjected to analysis to determine the suitability of adjusting the terminal CPU frequency for task 5. The start times of tasks 9, 10, and 11 are computed based on the ready time of their execution nodes, FT, and output transmission time of their predecessors (e.g., tasks 4, 5, and 6). If the start time of any of the tasks (9, 10, 11) equals the sum of FT and the corresponding output transmission time of task 5, task 5 will be deemed unsuitable. Otherwise, the CPU frequency for the execution of task 5 will be reduced.

E. Decreasing Makespan by DVFS

To enhance the overall solution that meets dual constraints, we propose an algorithm aimed at reasonably increasing the CPU frequency for specific tasks, thereby reducing makespan. Due to the task dependency, for any task that has multiple predecessors, the latest output arrival time of its predecessors will affect its start time. Therefore, if a predecessor v_i of v_k has the latest output arrival time, the suitable reduction of its execution time will efficiently shorten the finish time of v_k. Meanwhile, if the "latest" predecessor becomes the "earliest" predecessor by increasing CPU frequency, it is not efficient for entire performance optimization. Besides, considering that the above predecessor v_i may be also a predecessor of another task, so all successors of v_i can be considered to determine the appropriate execution time of v_i. The primary procedure is outlined as follows: 1) ECAO identifies tasks executed on the mobile terminal suitable for a CPU frequency increase; 2) subsequently, these identified tasks are sorted to establish their scheduling priority; and 3) ECAO incrementally raises the CPU frequency for each task in the determined order, based on its respective completion time.

During the initial step, ECAO computes time differences to determine which tasks are amenable to adjustment. For a task v_i executed on the mobile terminal, v_i is considered suitable for CPU frequency adjustment if the start times of all successor tasks are equal to the output arrival time of v_i; otherwise, it is deemed unsuitable for a frequency increase. In the second step, the scheduling priority for task v_i is determined by

$$P_i^3 = \frac{T_i^p - T_r^i}{E_i^r - E_i^p} \tag{18}$$

where E_i^p and E_i^r denote the energy consumption of terminal execution with the original and maximal CPU frequency for

Algorithm 4 Decreasing Makespan by DVFS

Input: Current offloading solution, energy constraint.
Onput: Improved offloading decision, CPU frequency for terminal tasks.

if the current energy consumption $\leq$ constraint
  for each v_i in the terminal processing queue S_q
    for each v_j in succ(i)
      if $(\mathrm{FT}_i + \mathrm{TT}(i,j))! = T_j^s$
        v_i is unsuitable and added to S_u;
        Break;
      end if
    end for
  end for
  Generate suitable task set $S_s = S_q - S_u$
  Calculate priority for each task in S_s by Eq. (18);
  Establish the priority queue;
  for each v_i in the priority queue
    Calculate the CPU frequency S_0^i by Eq. (20);
    Update the current energy consumption;
    if the current energy consumption $\leq$ constraint
      Retain the solution of step 14;
    end if
  end for
end if

v_i, respectively, and T_i^p and T_i^r are correspondingly the task finish time of v_i with the original and maximal CPU frequency.

In the third step, ECAO analyzes the specific finish time for each suitable task to obtain the corresponding CPU frequency. For each $v_j \in \mathrm{succ}(i)$, we define $\mathrm{FT}_i + \mathrm{TT}(i,j) - \mathrm{FT}_k - \mathrm{TT}(k,j)$ as a scaling duration in which task k has the output arrival time earlier than v_i and later than other predecessors of v_j. Meanwhile, v_k is a predecessor of v_j. The scaling duration is utilized to control the CPU frequency so as to maximize the efficiency. In fact, if a suitable local task is allocated with the maximal CPU frequency, the terminal energy consumption may be wasteful. For a task, the slowest predecessor determines its finish time. Once the suitable local task is not the slowest predecessor, a further increase in the CPU frequency is not helpful to decrease the successor finish time. Hence, we can derive the following inequality:

$$\frac{L_i}{S_1^i} + \min\{\mathrm{FT}_i + \mathrm{TT}(i,j) - \mathrm{FT}_k - \mathrm{TT}(k,j)\} \geq \frac{L_i}{S_1}. \tag{19}$$

When (19) satisfies the equality condition, the efficiency of CPU frequency scaling is maximized. Therefore, the updated CPU frequency for v_i can be obtained as

$$S_1^i = \frac{L_i}{\frac{L_i}{S_1} - \min\{\mathrm{FT}_i + \mathrm{TT}(i,j) - \mathrm{FT}_k - \mathrm{TT}(k,j)\}}. \tag{20}$$

The details of the above method is elaborated in Algorithm 4.

As shown in Fig. 4, we assume that task 5 resides within the terminal processing queue. The set succ(5), i.e., tasks 9, 10 and 11 will be analyzed so as to determine whether task 5 is suitable to raise terminal CPU frequency. The start time of tasks 9, 10, and 11 can be derived based on the ready instant of their execution nodes as well as FT and output transmission time of their predecessors. If the start time of any task within the set (9, 10, 11) does not align with the sum of FT and the corresponding output transmission time of task 5, task 5

is deemed unsuitable. Conversely, if the start time satisfies the condition, the CPU frequency for task 5's execution will be increased. Equation (20) is instrumental in determining a reasonable increment of CPU frequency for task 5. Notably, the output arrival time from task 5 to task 9 should not exceed that from task 4 to task 9.

Theorem 2: The overall time complexity of ECAO is $O(EN^2H + NH^2 + NMH + N^2M)$.

Proof: The time complexity of ECAO depends on the number of tasks N, the number of edges E, the number of execution nodes M and the variable H. Specifically, step 1 of Algorithm 1 has a time complexity of $O(N^2)$, and step 2 has a cyclic operation of N times. Besides, the time complexity of step 3 is $O(HN + HE)$, and that of step 4 is $O(H^2)$. For steps 5–16 in Algorithm 1, its time complexity is $O(HNE + HM)$). So, Algorithm 1 has an $O(EN^2H + NH^2 + NMH)$ time complexity. For Algorithm 2, steps 2 and 11 have the same time complexity of $O(N^2)$. For steps 4–10, its time complexity is $O(NM)$. Steps 13–15 has the time complexity of $O(N^2M)$. Thereby, Algorithm 2 has a time complexity of $O(N^2M)$. For Algorithm 3, steps 2–9 has a time complexity of $O(NE)$. The time complexity of steps 10–12 is $O(N^2)$, and steps 13–15 has a time complexity of $O(N)$. Consequently, the time complexity of Algorithm 3 is $O(N^2 + NE)$. Similar to Algorithm 3, the time complexity of Algorithm 4 is also $O(N^2+NE)$. Therefore, considering the time complexity of the four algorithms, we derive the overall time complexity of ECAO as $O(EN^2H + NH^2 + NMH + N^2M)$. ■

V. Performance Evaluation

In this section, we present the performance evaluation of ECAO, along with three existing approaches: 1) the genetic algorithm (GA); 2) TBTOA [21]; and 3) TOS [15]. Both TBTOA and TOS aim to optimize makespan while satisfying terminal energy consumption constraint. The performance metrics considered include the makespan, terminal energy consumption and service success ratio. A requested service has the constraints of service cost and terminal energy consumption. If the dual constraints are met, it is considered that the offloading solution is successful.

A C++ simulator is implemented to generate solutions for ECAO, GA, and TBTOA in response to a given service request. The simulator is capable of deriving makespan, terminal energy consumption, and service cost for each scheme. The simulation is executed 20 times for each test, incorporating different computation load distributions. The final result is the average value derived from the test results.

A. Experimental Setting

In the simulated edge–cloud collaboration system, transmission links in the local area network, metropolitan area network and wide area network have different transmission rates. Specifically, in the wireless local area network, the transmission rates between the mobile terminal and the neighboring edge server are calculated following (3) and (4). Besides, the channel gain follows the free-space path loss model $G_u = A_d(3 \times 10^8/4\pi f_c d)^{\mathrm{PL}}$ and $G_d = 0.4G_u$, where A_d denotes the antenna gain, f_c denotes the carrier frequency, d represents the distance between the mobile terminal and the access point, and PL denotes the pass loss exponent [16]. In the metropolitan area network, the transmission rate between two edge servers is set to 22 Mb/s. In the wide area network, the uploading rate between the mobile terminal and the central cloud is set to 3 Mb/s, and the corresponding downloading rate is 12 Mb/s. Besides, the transmission rate between the central cloud and an edge server is set to 12 Mb/s.

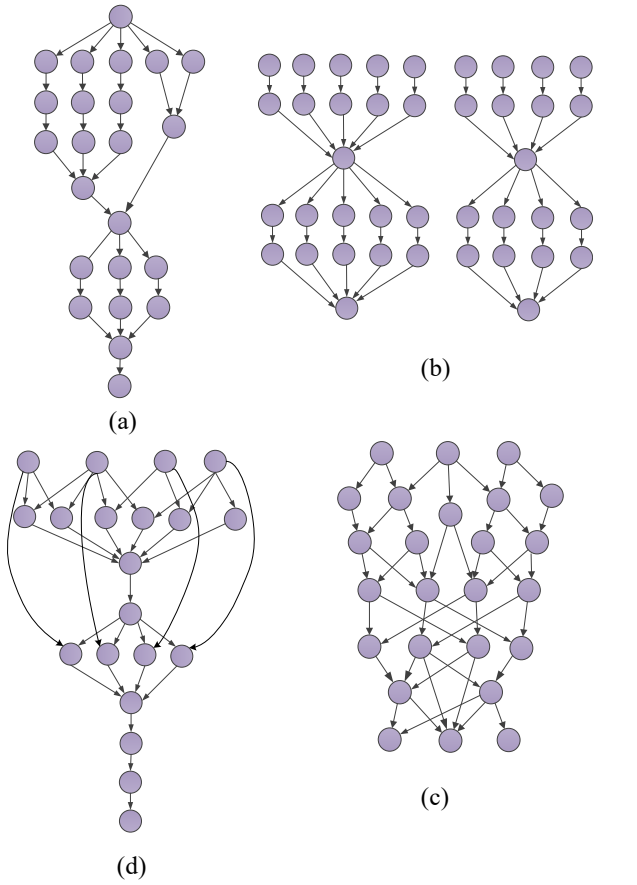

Fig. 5. Task graphs for performance evaluation. (a) First task graph (ER). (b) Second task graph (LIGO). (c) Third task graph (montage). (d) Fourth task graph (customized).

As shown in Fig. 5, we use four task graphs to evaluate performance. The first task graph is derived from ER application which is a representative real HDT application. The main functions of ER include preprocessing, generating feature matrices, feature selection based on clustering, fusion and recognition [5]. The second task graph is derived form a real DAG application LIGO [42], and the third task graph is derived form another real DAG application Montage [43]. Generally, the task graph with more complex task dependency is more difficult to deal with. The fourth task graph is artificially designed to generate different task dependency and parallelism from the other task graphs, and it is called Customized. In our simulation experiment, the service is provided by four nodes, including a central cloud and three edge servers. In GA, the initial population size is set to 50 [40]. The number of evolutionary iterations is set to 200 to meet

TABLE II
PARAMETERS IN EXPERIMENTS

Parameters	Value
The processing speed of central cloud	4.0 GHz
The processing speed of edge server 1	2.0 GHz
The processing speed of edge server 2	2.3 GHz
The processing speed of edge server 3	2.5 GHz
The local normal processing speed	1.5 GHz
The local minimal processing speed	0.5 GHz
The local maximal processing speed	3.5 GHz
Data size of one output of a task	[20, 1600] KB
Computation load of a task	[100, 2000] Mcycle
The transmission power of the access point	1 W
The uploading power of the mobile terminal	0.15 W
The downloading power of the mobile terminal	0.03 W
The mobile terminal CPU parameter δ	0.089×10^{-9}
The bandwidth W	2.6 MHz
The noise power	10^{-10} W
The antenna gain A_d	4.11
The carrier frequency f_c	915 MHz
The distance d	20
The pass loss exponent PL	3
The service cost per unit time of central cloud	250
The service cost per unit time of edge server 1	100
The service cost per unit time of edge server 2	120
The service cost per unit time of edge server 3	150

the requirement of being real time of decision making, and the specific parameters are illustrated in Table II.

B. Experimental Results and Analysis

Fig. 6 shows the makespan of GA, TBTOA, and TOS, and the proposed ECAO with different computation loads of the four task graphs. When the data size of each application maintains unchanged, the makespan of GA, TBTOA, TOS and ECAO shows an increasing trend with the increase of the total computation load for the four task graphs. As the total computation load increases, the more execution time will be expended so that the corresponding makespan rises. For the first task graph, the constraints of energy consumption and service cost are set to 2000 mJ and 2000. In all tests of the first task graph, ECAO has less makespan than GA, TBTOA, and TOS. Specifically, the makespan of ECAO is averagely 43%, 14%, and 15% less than GA, TBTOA, and TOS, respectively. For the second task graph, the constraints of energy consumption and service cost are set to 5000 mJ and 2800. In all tests of the second task graph, ECAO has less makespan than all comparison schemes. Specifically, the makespan of ECAO is averagely 26%, 11%, and 11% less than GA, TBTOA, and TOS, respectively. For the third task graph, the constraints of energy consumption and service cost are set to 2200 mJ and 2000. In all tests of the third task graph, ECAO has better performance in terms of makespan, and the makespan of ECAO is averagely 25%, 7%, and 8% less than GA, TBTOA, and TOS, respectively. For the fourth task graph, the constraints of energy consumption and service cost are set to 3000 mJ and 2500. In all tests of the fourth task graph, ECAO has less makespan than all comparison schemes. Specifically, the makespan of ECAO is averagely 32%, 22%, and 20% less than GA, TBTOA, and TOS, respectively. To summarize, ECAO achieves less makespan compared to GA, TBTOA, and TOS for all test task graphs, because the rescheduling algorithms in ECAO effectively reduce makespan.

Fig. 7 shows the average terminal energy consumption of GA, TBTOA, TOS, and the proposed ECAO, and the four task graphs have different computation loads and energy consumption constraints. There are two tests on each task graph. In the first test, the total computation load of each task graph is set to 15 000, 30 000, 10 000, and 22 000 (Mcycle), respectively, and the service cost constraint is set to 2000, 1800, 2000, and 2600 correspondingly. The total computation load of each task graph in the second test is greater than the first test by 2000 Mcycle. In the second test, the service cost constraint of each task graph is set to 2000, 2200, 2200, and 2600, respectively. In all tests, the average terminal energy consumption of ECAO is less than or equal to TBTOA and TOS. Specifically, ECAO achieves averagely 9% and 22% reduction of terminal energy consumption compared to TBTOA and TOS, respectively. Besides, for the second task graph, both TBTOA and TOS exceed the energy consumption constraint in the two tests. Although the average terminal energy consumption of GA is less than the other test schemes for the first, second and fourth task graphs, its effect on the third task graph shows the instability of performance due to the randomness of initial population and evolutionary operations. Moreover, the makespan of GA is significantly greater than the other test schemes. Hence, GA does not perform entirely well.

As shown in Fig. 8, the service success ratio with respect to energy consumption constraint is evaluated. The constraint of terminal energy consumption has a direct impact on service success ratio. When the data size, computation load and service cost constraint of each application remain unchanged, the service success ratio of ECAO shows a nondecreasing trend with the increase of the constraint of terminal energy consumption as well as GA, TBTOA, and TOS for the four task graphs. The reason of the above trend is that there will be equal or more feasible solutions in multiple tests as the energy consumption constraint becomes more relaxed. For the first task graph, the total computation load is set to 19 000 Mcycle, and the service cost constraint is set to 2000. In all tests of the first task graph, ECAO and GA achieve 100% service success ratio which is higher than that of TBTOA and TOS. Specifically, the service success ratio of ECAO is averagely 35 and 49 percentage points higher than TBTOA and TOS, respectively. For the second task graph, the total computation load is set to 32 000 Mcycle, and the service cost constraint is set to 2200. In all tests of the second task graph, ECAO and GA have the same service success ratio which is equal to or higher than that of TBTOA and TOS. Specifically, the service success ratio of ECAO is averagely 12 and 44 percentage points higher than TBTOA and TOS, respectively. For the third task graph, the total computation load is set to 10 000 Mcycle, and the service cost constraint is set to 2000. In all tests of the third task graph, the service success ratio of ECAO is equal to or higher than GA, TBTOA, and TOS. Specifically, the service success ratio of ECAO is averagely 89, 14 and 20 percentage points higher than GA, TBTOA, and TOS, respectively. For the fourth task graph,

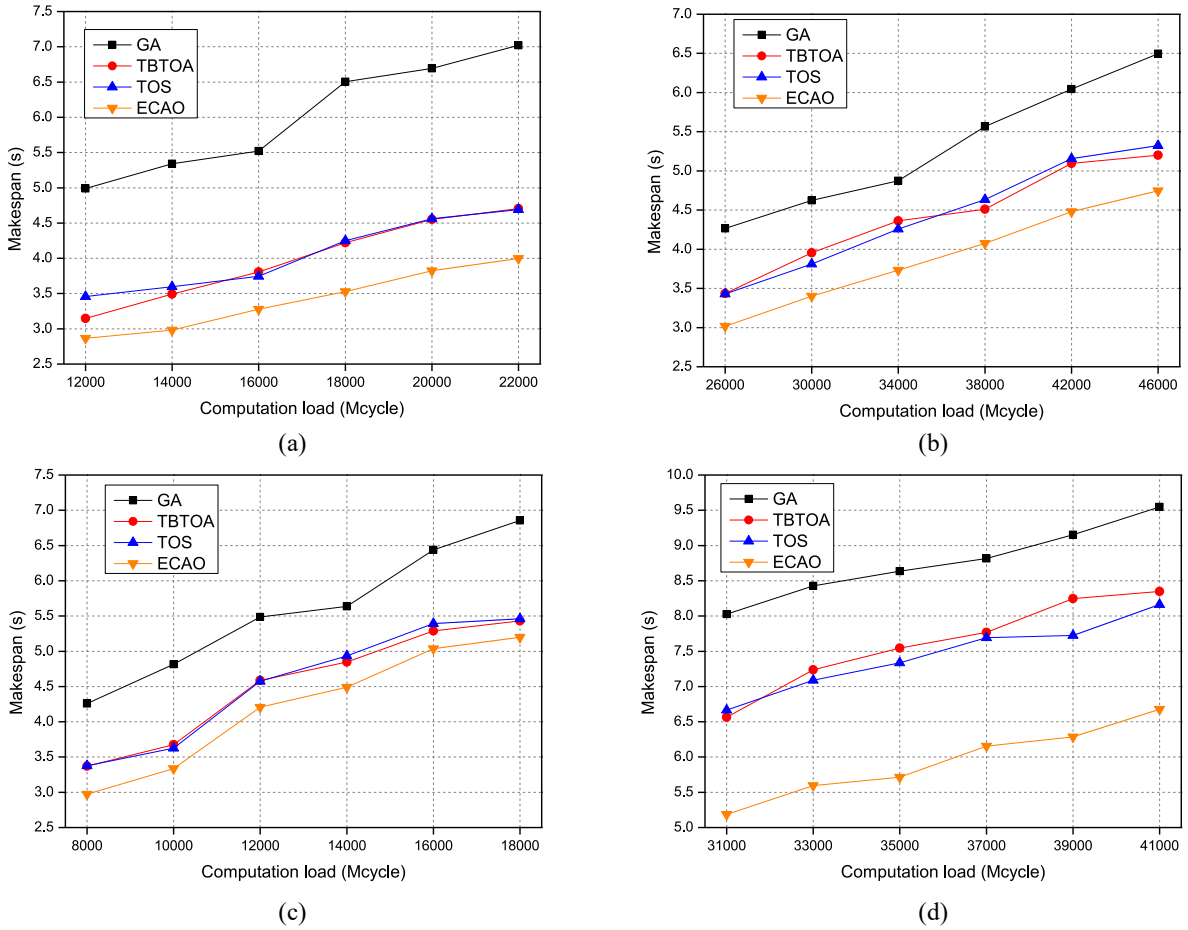

Fig. 6. Makespan with respect to computation load. (a) First task graph (ER). (b) Second task graph (LIGO). (c) Third task graph (montage). (d) Fourth task graph (customized).

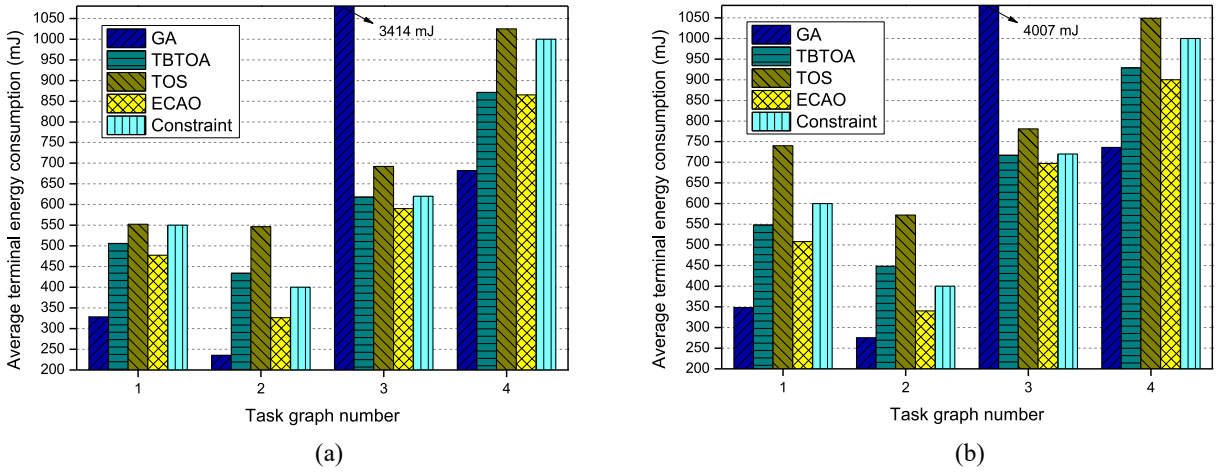

Fig. 7. Average terminal energy consumption. (a) First test on different task graphs. (b) Second test on different task graphs.

the total computation load is set to 30 000 Mcycle, and the service cost constraint is set to 2600. In all tests of the fourth task graph, ECAO and GA achieve 100% service success ratio which is higher than that of TBTOA and TOS. Specifically, the service success ratio of ECAO is averagely 28 and 27 percentage points higher than TBTOA and TOS, respectively. To summarize, ECAO performs better performance in terms of service success ratio compared to GA, TBTOA, and TOS in general, because the rescheduling algorithms in ECAO effectively optimize terminal energy consumption even if the constraint is relatively tight.

Fig. 9 shows the service success ration of GA, TBTOA, TOS, and the proposed ECAO with different service cost constraints of the four task graphs. When the data size,

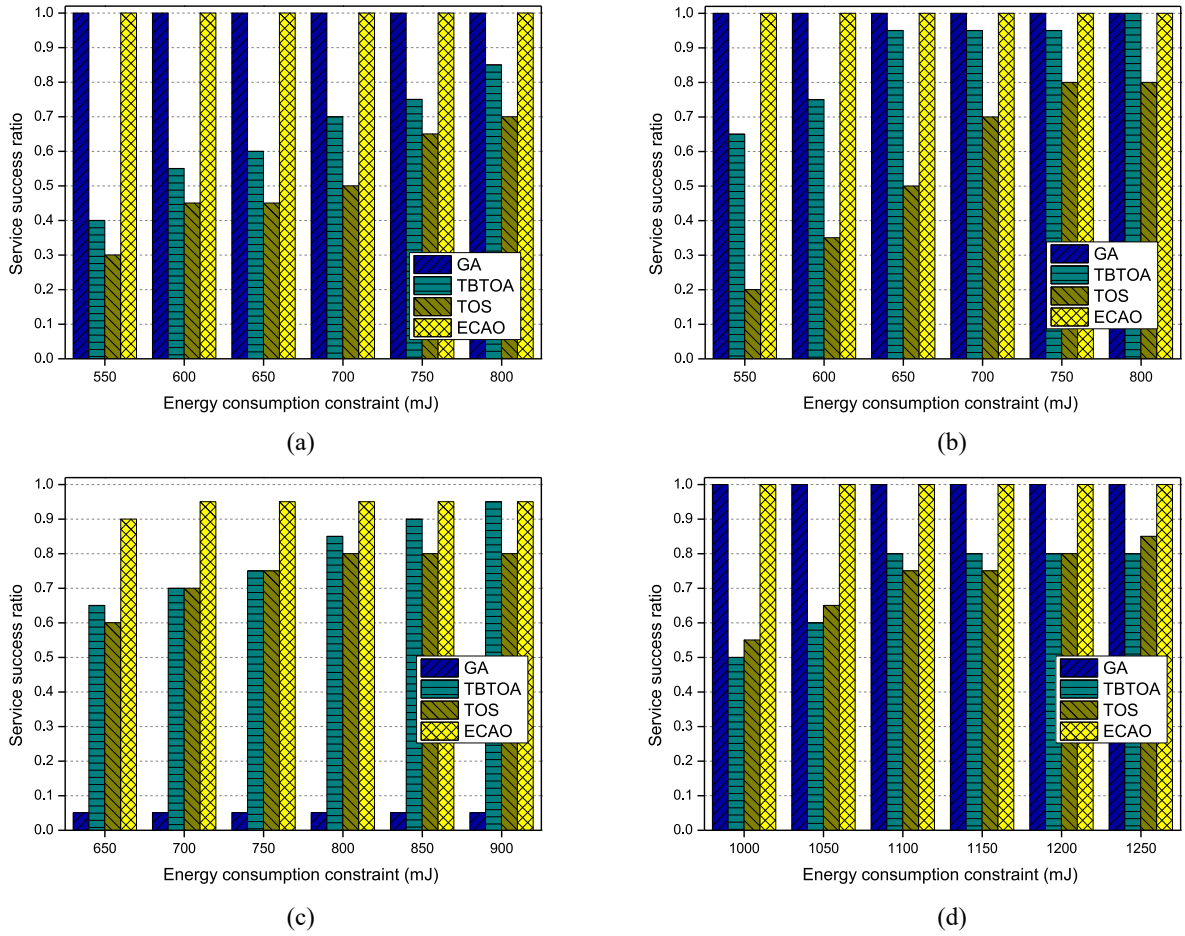

Fig. 8. Service success ratio with respect to energy consumption constraint. (a) First task graph (ER). (b) Second task graph (LIGO). (c) Third task graph (montage). (d) Fourth task graph (customized).

computation load and energy consumption constraint of each application remain unchanged, the service success ratio of ECAO shows a nondecreasing trend with the increase of the service cost constraint as well as GA, TBTOA, and TOS for the four task graphs. For the first task graph, the total computation load is set to 18 000 Mcycle, and the energy consumption constraint is set to 2200 mJ. In all tests of the first task graph, ECAO and GA achieve 100% service success ratio which is higher than that of TBTOA and TOS. Specifically, the service success ratio of ECAO is averagely 74 and 47 percentage points higher than TBTOA and TOS, respectively. Besides, when the service cost constraints are 800 and 840, TBTOA generates no feasible solution. For the second task graph, the total computation load is set to 31 000 Mcycle, and the energy consumption constraint is set to 2000 mJ. In all tests of the second task graph, ECAO and GA have the same service success ratio which is equal to or higher than that of TBTOA and TOS. Specifically, the service success ratio of ECAO is averagely 61 and 46 percentage points higher than TBTOA and TOS, respectively. In addition, the service success ratio of TBTOA and TOS is lower than 90% when the service cost constraint is less than or equal to 1650. For the third task graph, the total computation load is set to 10 000 Mcycle, and the energy consumption constraint is set to 2000 mJ. In all tests of the third task graph, the service success ratio of ECAO is equal to or higher than TBTOA and TOS, and it is close to that of GA. Specifically, the service success ratio of ECAO is averagely 34 and 36 percentage points higher than TBTOA and TOS, respectively. For the fourth task graph, the total computation load is set to 30 000 Mcycle, and the energy consumption constraint is set to 5000 mJ. In all tests of the fourth task graph, ECAO and GA achieve 100% service success ratio which is higher than that of TBTOA and TOS. Specifically, the service success ratio of ECAO is averagely 62 and 49 percentage points higher than TBTOA and TOS, respectively. To summarize, ECAO has higher service success ratio compared to TBTOA and TOS for most test constraints. Although the service success ratio of GA is very close to ECAO, the makespan of GA is significantly greater than ECAO. This is due to the fact that the rescheduling algorithm for service cost optimization in ECAO effectively adjusts service cost.

VI. Conclusion

In this article, the problem of dependent task offloading for HDT applications in the edge–cloud collaboration system is studied. We propose an ECAO scheme, called ECAO, which contains an initial offloading algorithm and three rescheduling algorithms. ECAO utilizes special task paths,

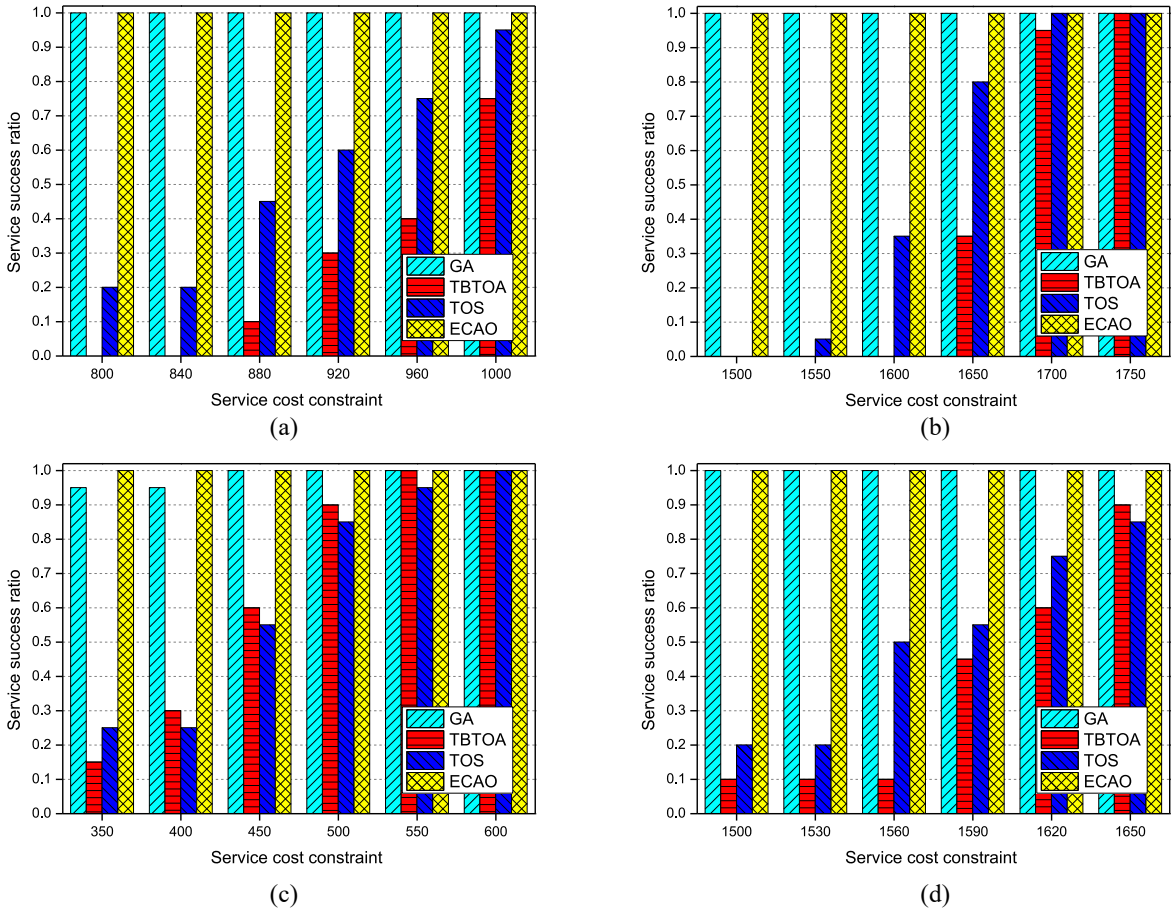

Fig. 9. Service success ratio with respect to service cost constraint. (a) First task graph (ER). (b) Second task graph (LIGO). (c) Third task graph (montage). (d) Fourth task graph (customized).

interdependency among tasks and DVFS to generate the offloading solution. The experimental results show that the ECAO scheme significantly decreases makespan and improves service success ratio compared to counterparts. In our future work, we may further investigate joint bandwidth allocation and computation offloading for HDT applications in the edge–cloud collaboration system for improving quality of services and meanwhile well balancing the service cost and terminal energy consumption.

REFERENCES

[1] S. D. Okegbile, J. Cai, D. Niyato, and C. Yi, "Human digital twin for personalized healthcare: Vision, architecture and future directions," *IEEE Netw.*, vol. 37, no. 2, pp. 262–269, Mar./Apr. 2023.

[2] Y. Wu, K. Zhang, and Y. Zhang, "Digital twin networks: A survey," *IEEE Internet Things J.*, vol. 8, no. 18, pp. 13789–13804, Sep. 2021.

[3] S. D. Okegbile, J. Cai, H. Zheng, J. Chen, and C. Yi, "Differentially private federated multi-task learning framework for enhancing human-to-virtual connectivity in human digital twin," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 11, pp. 3533–3547, Nov. 2023.

[4] J. Chen, C. Yi, S. D. Okegbile, J. Cai, and X. S. Shen, "Networking architecture and key supporting technologies for human digital twin in personalized healthcare: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 26, no. 1, pp. 706–746, 1st Quart., 2024.

[5] L. Chen, K. Wang, M. Li, M. Wu, W. Pedrycz, and K. Hirota, "K-means clustering-based kernel canonical correlation analysis for multimodal emotion recognition in human-robot interaction," *IEEE Trans. Ind. Electron.*, vol. 70, no. 1, pp. 1016–1024, Jan. 2023.

[6] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 7944–7956, Aug. 2019.

[7] Y. Wang, J.-T. Zhou, and X. Song, "A utility game driven QoS optimization for cloud services," *IEEE Trans. Services Comput.*, vol. 15, no. 5, pp. 2591–2603, Sep./Oct. 2022.

[8] D. Wu, R. Bao, Z. Li, H. Wang, H. Zhang, and R. Wang, "Edge–cloud collaboration enabled video service enhancement: A hybrid human-artificial intelligence scheme," *IEEE Trans. Multimedia*, vol. 23, pp. 2208–2221, Mar. 2021.

[9] J. Chen et al., "A revolution of personalized healthcare: Enabling human digital twin with mobile AIGC," *IEEE Netw.*, early access, Feb. 16, 2024, doi: 10.1109/MNET.2024.3366560.

[10] M. E. Miller and E. Spatz, "A unified view of a human digital twin," *Human-Intell. Syst. Integr.*, vol. 4, pp. 23–33, Mar. 2022.

[11] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, "Low-latency federated learning and blockchain for edge association in digital twin empowered 6G networks," *IEEE Trans. Ind. Informat.*, vol. 17, no. 7, pp. 5098–5107, Jul. 2021.

[12] H. Guo, Y. Wang, J. Liu, and C. Liu, "Multi-UAV cooperative task offloading and resource allocation in 5G advanced and beyond," *IEEE Trans. Wireless Commun.*, vol. 23, no. 1, pp. 347–359, Jan. 2024.

[13] X. Chen, S. Hu, C. Yu, Z. Chen, and G. Min, "Real-time offloading for dependent and parallel tasks in cloud-edge environments using deep reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 3, pp. 391–404, Mar. 2024.

[14] C. Feng, P. Han, X. Zhang, Q. Zhang, Y. Liu, and L. Guo, "Dependency-aware task reconfiguration and offloading in multi-access edge cloud networks," *IEEE Trans. Mobile Comput.*, early access, Feb. 1, 2024, doi: 10.1109/TMC.2024.3360978.

[15] H. Wang et al., "Low-complexity and efficient dependent subtask offloading strategy in IoT integrated with multi-access edge computing," *IEEE Trans. Netw. Service Manag.*, vol. 21, no. 1, pp. 621–636, Feb. 2024.
[16] J. Yan, S. Bi, and Y. J. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Wireless Commun.*, vol. 19, no. 8, pp. 5404–5419, Aug. 2020.
[17] J. Liu, J. Ren, Y. Zhang, X. Peng, Y. Zhang, and Y. Yang, "Efficient dependent task offloading for multiple applications in MEC-cloud system," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2147–2162, Apr. 2023.
[18] C.-T. Mo, J.-H. Chen, and W. Liao, "Graph convolutional network augmented deep reinforcement learning for dependent task offloading in mobile edge computing," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2023, pp. 1–6.
[19] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading tasks with dependency and service caching in mobile edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 11, pp. 2777–2792, Nov. 2021.
[20] L. X. Nguyen, Y. K. Tun, T. N. Dang, Y. M. Park, Z. Han, and C. S. Hong, "Dependency tasks offloading and communication resource allocation in collaborative UAV networks: A metaheuristic approach," *IEEE Internet Things J.*, vol. 10, no. 10, pp. 9062–9076, May 2023.
[21] X. Lv, H. Du, and Q. Ye, "TBTOA: A DAG-based task offloading scheme for mobile edge computing," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2022, pp. 4607–4612.
[22] T. Tang, C. Li, and F. Liu, "Collaborative cloud-edge-end task offloading with task dependency based on deep reinforcement learning," *Comput. Commun.*, vol. 209, pp. 78–90, Sep. 2023.
[23] 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.
[24] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.
[25] K. Li, M. Tao, and Z. Chen, "Exploiting computation replication for mobile edge computing: A fundamental computation-communication tradeoff study," *IEEE Trans. Wireless Commun.*, vol. 19, no. 7, pp. 4563–4578, Jul. 2020.
[26] R. Malik and M. Vu, "Energy-efficient computation offloading in delay-constrained massive MIMO enabled edge network using data partitioning," *IEEE Trans. Wireless Commun.*, vol. 19, no. 10, pp. 6977–6991, Oct. 2020.
[27] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, and L. Li, "Delay-aware and energy-efficient computation offloading in mobile-edge computing using deep reinforcement learning," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 3, pp. 881–892, Sep. 2021.
[28] J. Chen, H. Xing, Z. Xiao, L. Xu, and T. Tao, "A DRL agent for jointly optimizing computation offloading and resource allocation in MEC," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17508–17524, Dec. 2021.
[29] B. Yang, X. Cao, J. Bassey, X. Li, and L. Qian, "Computation offloading in multi-access edge computing: A multi-task learning approach," *IEEE Trans. Mobile Comput.*, vol. 20, no. 9, pp. 2745–2762, Sep. 2021.
[30] Z. Cao, P. Zhou, R. Li, S. Huang, and D. Wu, "Multiagent deep reinforcement learning for joint multichannel access and task offloading of mobile-edge computing in industry 4.0," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6201–6213, Jul. 2020.
[31] Y. Ren, Y. Sun, and M. Peng, "Deep reinforcement learning based computation offloading in fog enabled industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 17, no. 7, pp. 4978–4987, Jul. 2021.
[32] X. Wang, Z. Ning, and S. Guo, "Multi-agent imitation learning for pervasive edge computing: A decentralized computation offloading algorithm," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 2, pp. 411–425, Feb. 2021.
[33] X. Deng, J. Yin, P. Guan, N. N. Xiong, L. Zhang, and S. Mumtaz, "Intelligent delay-aware partial computing task offloading for multi-user industrial Internet of Things through edge computing," *IEEE Internet Things J.*, vol. 10, no. 4, pp. 2954–2966, Feb. 2023.
[34] C. Yi and J. Cai, "Two-stage spectrum sharing with combinatorial auction and Stackelberg game in recall-based cognitive radio networks," *IEEE Trans. Commun.*, vol. 62, no. 11, pp. 3740–3752, Nov. 2014.
[35] A. J. Goldsmith, *Wireless Communication*. Cambridge, U.K.: Cambridge Univ. Press, 2005.
[36] C. Yi, S. Huang, and J. Cai, "An incentive mechanism integrating joint power, channel and link management for social-aware D2D content sharing and proactive caching," *IEEE Trans. Mobile Comput.*, vol. 17, no. 4, pp. 789–802, Apr. 2018.
[37] C. Yi, J. Cai, and Z. Su, "A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications," *IEEE Trans. Mobile Comput.*, vol. 19, no. 1, pp. 29–43, Jan. 2020.
[38] F. Sun et al., "Cooperative task scheduling for computation offloading in vehicular cloud," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 11049–11061, Nov. 2018.
[39] K. D. Vogeleer, G. Memmi, P. Jouvelot, and F. Coelho, "The energy/frequency convexity rule: Modeling and experimental validation on mobile devices," in *Proc. Int. Conf. Parallel Process. Appl. Math.(PPAM)*, 2013, pp. 793–803.
[40] Z. Zhu, G. Zhang, M. Li, and X. Liu, "Evolutionary multiobjective workflow scheduling in cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1344–1357, May 2016.
[41] M. R. Gary and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1979.
[42] I. J. Taylor, E. Deelman, D. B. Gannon, and M. Shields, *Workflows for e-Science.*: Springer, 2007.
[43] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good, "The cost of doing science on the cloud: The montage example," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2008, pp. 1–12.

Qiang Zhang received the Ph.D. degree from the Department of Computer Science and Technology, Harbin Institute of Technology, Harbin, China, in 2016.

He is an Assistant Professor with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China. His research interests include cloud computing, edge computing, and Internet of Things.

Yuye Yang received the B.S. degree from the School of Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2022, where he is pursuing the M.S. degree with the College of Computer Science and Technology.

His main research interests include mobile edge computing, digital twin, stochastic optimization, resource management, and online learning.

Changyan Yi (Member, IEEE) received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, in 2018.

He is currently a Professor with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China. His research interests include stochastic optimization, mechanism design, game theory, queueing scheduling and machine learning with applications in resource management and decision making for edge computing and edge intelligence, mobile and human digital twin, ubiquitous intelligent network, and industrial cyber–physical system.

Samuel D. Okegbile (Member, IEEE) received the Ph.D. degree in computer engineering from the University of Pretoria, Pretoria, South Africa, in 2021.

He is currently a Postdoctoral Fellow with the Network Intelligence and Innovation Laboratory, Department of Electrical and Computer Engineering, Concordia University, Montreal, QC, Canada. His research interests include pervasive and mobile computing which includes various interesting topics in the human digital twin, the Internet of Things, data sharing, artificial intelligence, wireless communication networks, and blockchain.

Dr. Okegbile has received several awards, including the Horizon Postdoctoral Scholarship, the SENTECH Scholarship, and the University of Pretoria Doctoral Scholarship. He served as the Publication Chair for the 2023 Biennial Symposium on Communications. He is also a Regular Reviewer of some IEEE journals and conferences.

Jun Cai (Senior Member, IEEE) received the Ph.D. degree from the University of Waterloo, Waterloo, ON, Canada, in 2004.

From June 2004 to April 2006, he was a Natural Sciences and Engineering Research Council of Canada (NSERC) Postdoctoral Fellow with McMaster University, Hamilton, ON, Canada. From July 2006 to December 2018, he has been with the Department of Electrical and Computer Engineering, University of Manitoba, Winnipeg, MB, Canada, where he was a Full Professor and the NSERC Industrial Research Chair. Since January 2019, he has been a Full Professor and the PERFORM Centre Research Chair with the Department of Electrical and Computer Engineering, Concordia University, Montreal, QC, Canada. His current research interests include edge/fog computing, ehealth, radio resource management in wireless communication networks, and performance analysis.

Dr. Cai received the Best Paper Award from Chinacom in 2013, the Rh Award for outstanding contributions to research in applied sciences in 2012 from the University of Manitoba, and the Outstanding Service Award from IEEE Globecom 2010. He served as the Technical Program Committee (TPC) Co-Chair for IEEE GreenCom 2018; the Track/Symposium TPC Co-Chair for the IEEE VTC-Fall 2019, IEEE CCECE 2017, IEEE VTC-Fall 2012, IEEE Globecom 2010, and IWCMC 2008; the Publicity Co-Chair for IWCMC 2010, 2011, 2013, 2014, 2015, 2017, and 2020; and the Registration Chair for QShine 2005. He also served on the editorial board for IEEE INTERNET OF THINGS JOURNAL, *IET Communications*, and *Wireless Communications and Mobile Computing*.