A Novel Graph-Based Computation Offloading Strategy for Workflow Applications in Mobile Edge Computing

Xuejun Li, *Member, IEEE*, Tianxiang Chen, Dong Yuan, *Member, IEEE*, Jia Xu, *Member, IEEE*, and Xiao Liu, *Senior Member, IEEE*

Abstract—With the fast development of mobile edge computing (MEC), there is an increasing demand for running complex applications on the edge. These complex applications can be represented as workflows where task dependencies are explicitly specified. To achieve better Quality of Service (QoS), computation offloading is widely used in the MEC environment. However, many existing computation offloading strategies only focus on independent computation tasks but overlook the task dependencies. Meanwhile, most of these strategies are based on search algorithms which are often time-consuming and hence not suitable for many delay-sensitive complex applications in MEC. Therefore, a highly efficient graph-based strategy was proposed in our recent work but it can only deal with simple workflow applications with linear (namely sequential) structure. For solving these problems, a novel graph-based strategy is proposed for workflow applications in MEC. Specifically, this strategy can deal with complex workflow applications with nonlinear (viz. parallel, selective and iterative) structures. Meanwhile, the offloading decision plan with the lowest energy consumption of the end-device under deadline constraint can be found by using the graph-based partition technique. We have comprehensively evaluated our strategy on FogWorkflowSim platform for complex workflow applications. Extensive numerical results demonstrate that the end device's energy consumption can be effectively reduced by 7.81% and 9.51% compared with PSO and GA by the proposed strategy. Meanwhile, the strategy running time is 1% and 0.2% of PSO and GA, respectively.

Index Terms—Mobile Edge computing, workflow management, energy consumption, computation offloading, directed acyclic graph

1 Introduction

With the continuous improvement of the computing capacity of various smart end-devices, an increasing number of intelligent applications are deployed on mobile end-devices such as smart logistics. Meanwhile, massive computation requests submitted by the end-devices can be offloading to cloud data centers. However, public networks with limited bandwidth will cause significant delays, which is unacceptable for many delay-sensitive applications. Nowadays, mobile edge computing (MEC) has been widely used to provision computing resources from the network edge to the end-device in order to reduce response delay [1]. Computation tasks on the end-devices can be offloaded to the edge servers for execution through low-cost and high-bandwidth transmission such as the 5G and WIFI networks [2], [3]. Computation offloading plays a key role in effectively improving the Quality of Service (QoS) of MEC-based applications by reducing the response delay and the energy consumption of end-devices [4], [5].

Given the success of MEC, there is an increasing demand for running complex applications on the edge. For example, in the UAV (Unmanned Aerial Vehicle) based smart delivery system, there are many complex applications such as dynamic route planning, obstacle detection and face recognition [6]. At the same time, UAVs are limited by their computing power and battery life so that they are unable to execute computation-intensive tasks as mentioned above. Fortunately, the UAV's energy consumption and task response time can be effectively reduced by the computation offloading technology in the MEC environment [7]. Most of complex applications in the real-world can be represented by workflows where task dependencies are explicitly specified [8]. Business workflows usually focus on the modeling of controlflow oriented business processes. Scientific workflows aim to model large-scale, distributed, data-intensive and computation-intensive scientific processes [9]. This paper focuses on data-centric scientific workflows. Specifically, the task dependencies expressed as the input data of each task or whether it is executed depends on the execution result of the previous task. Most of research works are either based on independent tasks or simple task composition where tasks are executed in a sequential manner [10], [11].

The graphical description for scientific workflow is an intuitive way, such as Petri net and Directed Acyclic Graph (DAG). Petri net have the function of simple graphical

- *Xuejun Li, Tianxiang Chen, and Jia Xu are with the School of Computer Science and Technology, Anhui University, Hefei, Anhui 230093, China. E-mail: xjli@ahu.edu.cn, biyisi_96@qq.com, xujia@stu.ahu.edu.cn.*
- *Dong Yuan is with the School of Electrical and Information Engineering, University of Sydney, Sydney, NSW 2006, Australia. E-mail: dong.yuan@sydney.edu.au.*
- *Xiao Liu is with the School of Information Technology, Deakin University, Geelong, VIC 3220, Australia. E-mail: xiao.liu@deakin.edu.au.*

Manuscript received 2 July 2021; revised 16 Apr. 2022; accepted 25 May 2022. Date of publication 3 June 2022; date of current version 10 Apr. 2023.
This work was supported by the National Natural Science Foundation of China under Grants 61972001 and 62076002.
(Corresponding author: Xiao Liu.)
Digital Object Identifier no. 10.1109/TSC.2022.3180067

description and interpretation ability, but make graphs too redundant. DAG is one of the most general representation models for workflow tasks, and has the advantage of being easier to use and more intuitive [12]. In this paper, the workflow applications defined in DAG and also can be adapted to the workflow applications defined in other models. The Graph4Edge-Nonlinear strategy can be executed only by converting other workflow models into DAGs, and many existing research studies have already demonstrated the feasibility of model conversion, such as [13], [14].

Generally speaking, all real-world applications can be represented by a mix of linear (namely sequential) and nonlinear (viz. parallel, selective, and iterative) structures workflow [15]. It needs the computation offloading strategy should be able to deal with linear and nonlinear structures simultaneously. The greedy type of strategies have been widely used to obtain a feasible solution in a short time, but they cannot produce optimal offloading decisions [16]. In order to improve the quality of the decisions, many studies employed search algorithms such as particle swarm optimization (PSO) and genetic algorithm (GA) to search for the optimal offloading decision by an iterative process, which could produce significant time overhead [17], [18], [19], [20]. At present, most computation offloading strategies are either simple but not good enough, or they are too time-consuming to be unsuitable for delay-sensitive complex applications [21].

To address the above issues, a novel computation offloading strategy using graph partition technology is proposed in this paper for workflow applications in MEC. To distinguish with our previous preliminary work which can only deal with linear workflow structures [22], we name the previous strategy Graph4Edge-Linear, name the new strategy proposed in this paper Graph4Edge-Nonlinear. Our proposed strategy considers the influence of the complex task dependencies on the computation offloading decisions, and the end device's energy consumption is optimized effectively under the given deadline constraints. Please note that the energy consumptions of edge servers are not considered in this paper. This is because edge servers are usually connected to the power grid, and hence their energy consumptions are not regarded as limiting factors in a MEC environment.

Specifically, the novel contributions of this paper are summarized as follows:

1) A novel nonlinear workflow model for complex MEC-based applications is proposed. The model is based on WDG (Workflow Dependency Graph) which considers both complex task dependencies and the objective of reducing the end-device's energy consumption.
2) We propose a novel graph-based computation offloading strategy named Graph4Edge-Nonlinear based on the WDG which can find the best computation offloading decision with the minimum end-device' energy consumption under the given deadline. Its performance is significantly better than popular search-algorithm based strategies.
3) Both a case study on a real-world UAV delivery system [23], [24] and extensive simulation experiments on the FogWorkflowSim platform [25] for MEC based workflow applications are presented. The experimental results demonstrate the effectiveness of our proposed strategy and its overall better performance than other representative strategies, especially in terms of strategy running time.

The rest of this paper is structured as follows: Section 2 introduces a motivating example on a MEC-based UAV delivery system. Section 3 presents some preliminaries for this study. Section 4 proposes our novel graph-based computation offloading strategy for workflow applications with nonlinear structures. Section 5 presents the evaluation results. Section 6 reviews the related works for computations offloading. Finally, Section 7 makes the conclusions and points out some future work.

2 Motivation Example and Problem Analysis

In this section, in order to describe the motivation of computation offloading in the MEC environment, an example of the MEC-based UAV last-mile delivery scenario is presented.

In the MEC-based UAV last-mile delivery system [23], there are various delay-sensitive applications such as dynamic flight route planning and autonomous obstacle avoidance for UAVs, pose and face recognition for receivers [26], [27]. In fact, because of the UAV's limited battery life and computing power, these computation-intensive tasks are not suitable for execution locally under the fast response and energy efficiency requirements. Therefore, computation offloading to the edge server is often required.

Here, we illustrate the computation offloading problem with a partial workflow of the whole UAV delivery process, namely the final parcel delivery workflow. The MEC-based UAV delivery system is conceptually divided into two layers. As shown in Fig. 1, the upper layer is the edge server layer, which consists of various edge servers. These edge servers can provide computing resources close to the end-device. The bottom layer is the final parcel delivery workflow which consists of a set of computation tasks with dependencies. In this paper, we simply refer temporal and data dependencies as task dependencies, and they can be explicitly specified using DAG as will be introduced in the next section. Specifically, as shown in Fig. 1, there are many computation-intensive tasks in the UAV last-mile delivery scenario, such as target detection, image segmentation, pose and face recognition. The processing of each target (namely the receiver) in the video frame is independent with each other. Hence, it can be regarded as parallel tasks in the workflow. The system needs to run multiple poses and face recognition tasks in parallel to ensure timely detection of the actual receiver from the crowd. Once the UAV detects the actual receiver, it will approach the receiver and begin to land and unload the parcel. More information about this UAV delivery system can be found in our previous works [23], [24].

Obviously, these real-time tasks are delay-sensitive, and fast response time is essential. According to different characteristics (such as task workload, data size and deadline constraints) of the computation tasks, some of them are offloaded to edge servers to achieve faster response time and

lower energy consumption [28]. However, computation offloading is a difficult decision-making problem.

While there are some existing strategies that are based on heuristic algorithms or search algorithms, they all have some limitations. For example, heuristic algorithms have the premature convergence issue so they may not be able to find the best computation offloading decision. While search algorithms such as PSO and GA can find the better decisions, they are usually very time-consuming and hence not suitable. Meanwhile, to the best of our knowledge, none of the existing strategies can effectively deal with complex task dependencies which can be represented by nonlinear workflow structures.

For solving the above issues, a novel graph-based strategy is proposed to solve the computation offloading problem in the MEC environment. Our proposed strategy can deal with nonlinear workflow structures and find the best computation offloading decision with the minimum energy consumption under the deadline constraint.

3 Preliminaries

Generally speaking, for the purpose of computation offloading, there are two kinds of computation tasks in the workflow, which are general tasks and local execution tasks. General tasks are those tasks which are executed either at end-device or edge server via computation offloading. Local execution tasks are those tasks that can only be executed on end-device due to the required input data is only available at the end-device and cannot be moved due to security restrictions, or some tasks which require user input at the end-device [22]. In this situation, edge servers cannot handle these tasks. In other words, these tasks must be processed on the end-device. These nodes can be expressed as $localSet$.

This paper uses the workflow dependency graph (WDG) to represent the workflow model and its task dependencies. WDG is a directed acyclic graph that is composed of workflow tasks with dependencies. Each task T_i of WDG contains three basic attributes x_i, y_i, z_i, which represent the energy consumption in different situations of the end-device.

In Fig. 2, the symbol $\rightarrow$ denotes that there is a dependency relationship between two task nodes. For example, the $T_i \rightarrow T_j$, indicates that T_i is the predecessor of T_j in the WDG. There are $T_1 \rightarrow T_2$, $T_2 \rightarrow T_3$, $T_2 \rightarrow T_5$, $T_3 \rightarrow T_4$, $T_5 \rightarrow T_6$, etc. T_1 points to T_2, which means there is a direct dependency between T_1 and T_2. We use the task T_1's output data as task T_2's input data. In addition, $\rightarrow$ is defined as having transitivity, where $T_i \rightarrow T_k \rightarrow T_j \Leftrightarrow T_i \rightarrow T_k \wedge T_k \rightarrow T_j \Leftrightarrow T_i \rightarrow T_j$.

The symbol $\nrightarrow$ indicates that there is no dependency between the two tasks, where $T_i \nrightarrow T_j$ means the T_i and T_j are disparate branches in WDG. For instance, we have $T_3 \nrightarrow T_5$, $T_4 \nrightarrow T_6$, etc. in Fig. 2.

Here, x_i means the energy consumption of the data transmission when decide offloading T_i to edge server. The data transmission between T_i and T_j (MB) indicated as $Comm(T_i, T_j)$. T_i's direct predecessor node is T_j. $Bandwidth$ is the data transfer speed of the computation tasks (Mbps). P_{trans} denotes the end-device's transmission power (W).

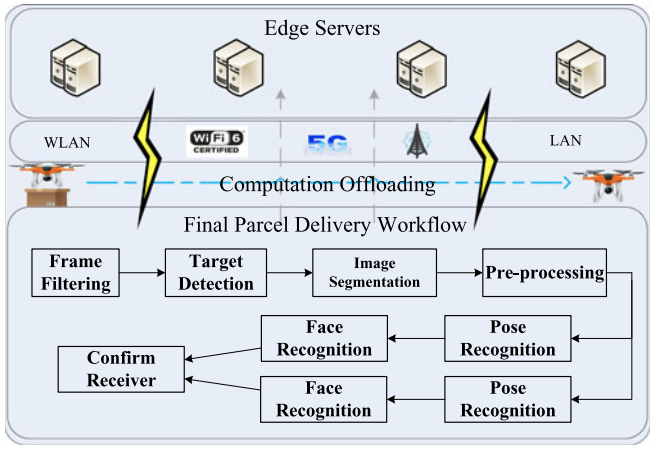


Fig. 1. Computation offloading for an example workflow in a MEC-based UAV delivery system.

$$x_i = \frac{Comm(T_i, T_j)}{Bandwidth} * P_{trans} \tag{1}$$

Where y_i is end-device's idle energy consumption when decide offloading T_i to edge server. The task T_i's workload (Megacycles) is denoted as l_i. The edge server's CPU frequency (GHz) is f_{edge}. The end-device's idle power (W) is P_{idle}.

$$y_i = \frac{l_i}{f_{edge}} * P_{idle} \tag{2}$$

The end-device's load energy consumption is denoted as z_i when T_i is executed on the end-device. The end-device's CPU frequency (GHz) and execution power (W) is denoted as f_{end} and P_{end} respectively.

$$z_i = \frac{l_i}{f_{end}} * P_{end} \tag{3}$$

In our previous work [22], the proposed method can convert the computation offloading problem of the linear WDG into the shortest path problem. The weight of the edge between two nodes is expressed as $w\langle T_i, T_j\rangle$, which is the end-device's energy consumption. At this time, the computation task T_i and T_j are decided offloading to edge server and the tasks between T_i and T_j are executed locally. All possible offloading decisions in WDG can be mapped to edges between different task nodes. The weight of each edge represents the energy consumption of end-devices. As mentioned before, the energy consumptions of edge servers are not considered as edge servers are usually connected to the power grid. There are two types of virtual nodes in the WDG, which are the start node T_s and end node T_e in problem of the shortest path. By adding T_s and T_e to the $localSet$, the energy consumption of returning the final result to the end-device can be calculated.

Fig. 3 demonstrates simple example to building an Energy consumption Transitive Graph (ETG) for a lineaer WDG where $localSet = \{T_3\}$. There are only two virtual nodes in graph, which are the start node with only in-edges and the end node with only out-edges. Add energy consumption edges for task nodes, and set the weight to the energy consumption required for offloading.

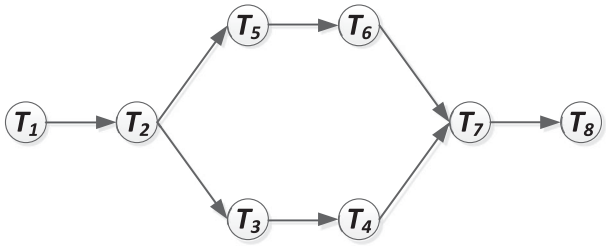


Fig. 2. An example workflow dependency graph.

According to the WDG, ETG an is constructed [18]. Specifically, we design a one-to-one mapping for all paths in the graph to all possible offloading decisions of the workflow, and the classic Dijkstra algorithm is used to find the shortest path in ETG. Since the complexity of graph-based algorithm is low, this strategy can efficiently obtain excellent results.

4 Graph-Based Minimum Energy Consumption Computation Offloading Strategy

We present the strategy of Graph4Edge-Nonlinear in this section. This strategy will find the best offloading decision with minimum energy consumption of the end-device under the given deadline. First in Section 4.1, we introduce our model definition and how to convert workflow structure to WDG. Then in Section 4.2, we discuss how to find the best offloading decision plan for complex WDG with Graph4Edge-Nonlinear. Finally, in Section 4.3, we use pseudo-code to describe the detailed process for our strategy and the discussion of its algorithm complexity is also presented.

4.1 Problem Formulation

We introduce the model definition in this section. Then the conversion from nonlinear workflow structure to WDG is described.

4.1.1 Model Definition

For the computation offloading purpose, the attributes of task T_i are defined as $x_i, y_i, z_i, flag_i, E_i$. $flag_i$ denotes the constraint whether T_i is local execution task or not. Specifically, $flag_i = 1$ means that T_i is a local execution task. Otherwise, the task T_i is sending to the edge server for execution. E_i denotes the T_i's execution energy consumption in end-device. The calculation method of E_i as follows:

$$E_i = \begin{cases} x_i + y_i, if flag_i = 0 \ and flag_{i-1} = 1; \\ x_i + z_i, if flag_i = 1 \ and flag_{i-1} = 0; \\ y_i, if flag_i = 1 \ and flag_{i-1} = 1; \\ z_i, if flag_i = 0 \ and flag_{i-1} = 0; \end{cases} \tag{4}$$

According to the offloading decision, the calculation method of E_i has two situations. When either of task T_i and T_{i-1} offloaded to edge server, the data transmission energy x_i appears between the two tasks. Otherwise, when both of the two tasks are offloaded or not offloaded, there is no data transmission energy between the two tasks. The end device's energy consumption y_i or z_i depends on offloading

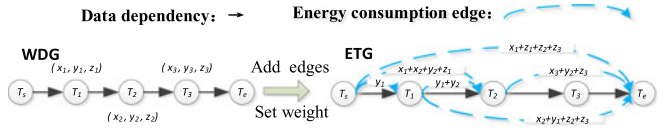


Fig. 3. An example of building an ETG for simple linear WDG.

decision plans of task T_i. The time consumption of each task T_i consists of transmission time T_i^{trans} and execution time T_i^{execu}. According to the above definition, the total time to complete all tasks is defined as follows:

$$Time_{\cos t} = \sum_{\{T_i | T_i \in WDG\}} (T_i^{trans} + \mathrm{T}_i^{execu}) \tag{5}$$

Then the destination function is defined as $MinEC$, which is calculated as follows:

$$\begin{aligned} MinEC &= \min \sum_{\{T_i | T_i \in WDG\}} (E_i) \\ s.t.\ Time_{cost} &\leq \mathrm{Deadline} \end{aligned} \tag{6}$$

The final optimization goal is to find the best computation offloading plan with the minimum energy consumption of the end-device under the deadline constraints.

As shown in Fig. 4, the WDG has a sub-branch within one block. To find the task offloading decision with minimum energy consumption in WDG, one branch is chosen randomly to construct the ETG which is called "main branch" (indicated as MB). The rest of branches are called "sub-branches" (indicated as SB). such as the $MB = \{T_1, T_2, T_3, T_4, T_7, T_8\}$ and $SB = \{T_5, T_6\}$ in Fig. 4. If the energy consumption of the SB in the block can be mapped to the weight of the MB, all offloading decesions can be represented on the MB. The weight of the SB in Fig. 4 is defined as $E_5 + E_6$.

The weights on the SB need to be taken into account when the weighted edge on the MB crosses the Block. At this point, the weight of SB depends on whether the predecessor node of the Block is offloading or not. Therefore, we need to discuss the in-block edge, out-block edge and normal edge separately. For the offloading decision that does not cross through the internal nodes of the Block, it will be easier to discuss their weights, so the Over-block edge is defined for consideration. To better present our problems and methods, here are some detailed definitions:

Block: In WDG, A block (denoted as B) is defined as a set of sub-branches which are forked from one task node and merged at another task node. A WDG with simple block $B = \{T_3, T_4, T_5, T_6\}$ is shown in Fig. 4.

In-block edge: In-block edge $e\langle T_i, T_j\rangle$ represents the edge begins with T_i preceding the block, and points to T_j in the block, for example, $e\langle T_1, T_3\rangle$, $eT_1, T_4\rangle$ in Fig. 4. Formally,

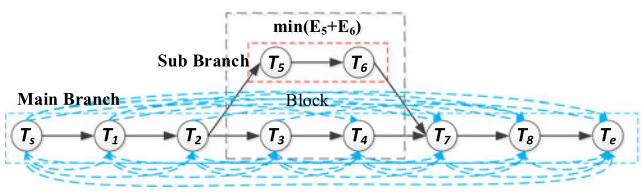


Fig. 4. An example of building ETG for single-block WDG.

$e\langle T_i, T_j\rangle$ is an in-block edge, where $\exists T_k \in WDG \wedge T_i \rightarrow T_k \wedge T_j \nleftrightarrow T_k$.

Out-block edge: Out-block edge $e\langle T_i, T_j\rangle$ represents the edge begins with T_i in the block, and points to T_j succeeding the block, for example, $e\langle T_3, T_8\rangle$, $e\langle T_4, T_8\rangle$ in Fig. 4. Formally, $e\langle T_i, T_j\rangle$ is an out-block edge, where $\exists T_k \in WDG \wedge T_i \nleftrightarrow T_k \wedge T_j \rightarrow T_k$.

Over-block edge: Over-block edge $e\langle T_i, T_j\rangle$ represents the edge crosses over the block. T_i is the task node which precedes the block and T_j is the task node which succeeds the block. For example, $e\langle T_1, T_8\rangle$, $e\langle T_2, T_8\rangle$ in Fig. 4. Formally, $e\langle T_i, T_j\rangle$ is an over-block edge, where $\exists T_k, T_h \in WDG \wedge T_i \rightarrow T_k \rightarrow T_j \wedge T_i \rightarrow T_h \rightarrow T_j \wedge T_h \nleftrightarrow T_k$.

Ordinary edge: An ordinary edge $e\langle T_i, T_j\rangle$ means that tasks between T_i and T_j when they are totally ordered, such as $e\langle T_1, T_2\rangle$, $e\langle T_3, T_4\rangle$, $e\langle T_7, T_8\rangle$ in Fig. 4. Formally, $e\langle T_i, T_j\rangle$ is an ordinary edge, where $\neg\exists T_k \in WDG \wedge ((T_i \rightarrow T_k \wedge T_k \nleftrightarrow T_j) \vee (T_i \nleftrightarrow T_k \wedge T_k \rightarrow T_j) \vee (T_h \in WDG \wedge T_h \nleftrightarrow T_k \wedge T_i \rightarrow T_h \rightarrow T_j \wedge T_i \rightarrow T_k \rightarrow T_j))$.

In this paper, we can define the weight of the ordinary edge as Eq. (7). For an edge $e\langle T_i, T_j\rangle$, we denote its weight as $\omega\langle T_i, T_j\rangle$, which is defined as the sum of the T_j's idle energy consumption, the transmission energy consumption of the T_i's successor and T_j, and the load energy consumption of the tasks between T_i and T_j, supposing that only T_i and T_j are all offloaded to edge server.

$$w(T_i, T_j) = y_j + x_{i+1} + x_j + \sum_{\{Tk \mid Tk \in WDG \wedge T_i \rightarrow T_k \rightarrow T_j\}} z_k \quad (7)$$

4.1.2 Convert Nonlinear Workflow to Complex WDG

The topology of workflow is represented by WDG. In the real-world, the structure of workflow contains four basic topology types, viz. sequential, parallel, selective and iterative structures [15]. Specifically, in this paper, we name the workflow only composed with the sequential structure as the linear workflow. If it contains the other three structures, we name the workflow as the nonlinear structure. Although the real-world workflow structures can be very complex with the mix of the four basic workflow structures, all of them can be converted to WDGs by a simple method are proposed in [29]. In [22], a strategy that can find minimum energy consumption of a simple linear structure workflow is proposed. However, the task dependencies in the remaining three workflow structures are much more complicated. Through the conversion process, any structures can be converted into multiple sequential structures. Fig. 5 shows how the three nonlinear basic workflow structures are converted to WDG.

As shown in Fig. 5a, three sequential structures are obtained by constructing three subtask instances for a parallel structure [29]. Firstly, add the virtual start node and end node. Secondly, the different paths to T_4 are constructed as separate branches. Finally, connect all paths to form a complete WDG. Figs. 5b and 5c show the conversion examples for the selective and iterative structures, respectively [8]. In Fig. 5b, still have to add two virtual start node and end node, then list three different path options to reach the T_5, shown as complex branch structure. The three cases of reaching T_5 are fully represented. In Fig. 5c, at least one passes through T_2 and at most two passes through the iterative loop. According to [8], it can be expressed as two branches in parallel.

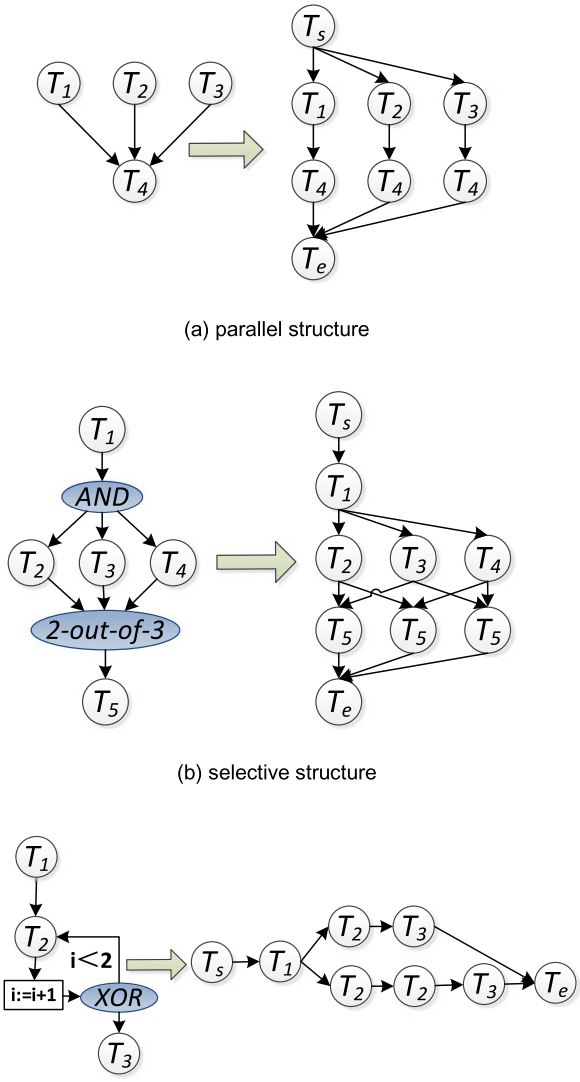


Fig. 5. Convert nonlinear workflow structures to WDG.

Any workflow is a combination of the four basic structures, and they can be converted into corresponding WDG models. In real-world workflow applications, WDGs with nonlinear structures are very common [30]. Due to the existence of nonlinear structures, the existing computation offloading strategy for linear structures cannot be used directly. For solving the problem of computation offloading for complex workflow applications with nonlinear structures, we propose the Graph4Edge-Nonlinear strategy. This strategy is able to optimize the end-device's energy consumption under the given deadline constraints.

4.2 Graph4Edge-Nonlinear for Complex WDG

In this subsection, we introduce our strategy implementation steps by single-block WDG firstly. Then, the Multi-block WDG strategy is introduced.

4.2.1 Single-Block WDG

In this section, the single-block WDG is analyzed as an example to describe the problem and detailed steps for Graph4Edge-Nonlinear based on above model definition.

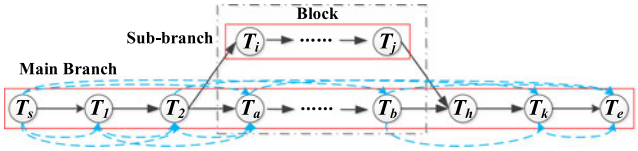


Fig. 6. The initial ETG of single-block WDG.

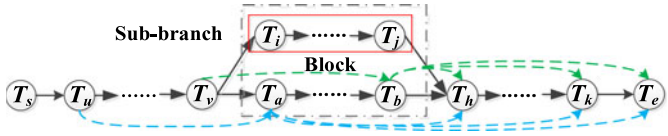


Fig. 7. Examples of in-block edges in two different situations.

The purpose of the Graph4Edge-Nonlinear strategy is to map the energy consumption of the possible offloading decision to the weight of the edge. Therefore, the shortest path of the linear workflow structure can be easily found by the optimal offloading strategy. $e\langle T_i, T_j\rangle$ means that tasks T_i and T_j are offloaded to the edge server, and tasks between T_i and T_j are executed locally. As a result, it is necessary to calculate the energy consumption of locally executed tasks which include transmission and idle energy consumption. In the single-block WDG, Eq. (7) is suitable for in-block edges and ordinary edges. But when $e\langle T_i, T_j\rangle$ is either out-block or over-block, Eq. (7) is no longer suitable for its weight calculation due to the tasks succeeding the block may have more than one task as their predecessor node. For example, the edge of $\langle T_3, T_8\rangle$ in Fig. 4 can be calculated by $w\ \langle T_3, T_8\rangle = x_4\ + x_8 + y_8 + z_4 + z_7$ according to Eq. (7). However, the offloaded situation of the sub-branch tasks T_5 and T_6 is not considered. Therefore, the obtained shortest path cannot represent the overall decision of WDG.

For the above reasons, the weight of $e\langle T_i, T_j\rangle$ is defined as follows:

$$w(T_i, T_j) = y_j + x_j + x^*_{i+1} + \sum_{\{T_k \mid T_k \in WDG \wedge T_i \to T_k \to T_j\}} z_k + \left(\sum_{\{T_l \mid T_l \in SB\}} E_{Tl}\right) S\min \quad (8)$$

In Eq. (8), $\left(\sum_{\{T_l \mid T_l \in \mathrm{SB}\}} E_{T_l}\right)_{S_{min}}$ means the minimum energy consumption of the tasks that are in the sub-branches of the block. x^*_{i+1} represents the transmission energy consumption of the task that directly depends on T_i. The out-block or over-block edge's shortest path length is equal to the task's minimum energy consumption by Eq. (8), i.e., $P_{min}\ \langle T_i, T_j\rangle = \sum_{\{T_k \mid T_k \in \mathrm{WDG} \wedge T_i \to T_k \to T_j\}} E_k$. Hence, in order to calculate the out- or over-block edge's weights, the offloaded strategy of sub-branch in single-block WDG is essential. For instance, the weight of edge $e\langle T_3, T_8\rangle$ in Fig. 4 is calculated as $w\ \langle T_3, T_8\rangle = x_4\ + x_8 + y_8 + z_4 + z_7 + (E_5 + E_6)_{S_{min}}$ where S_{min} is the best computation offloading decision of the SB.

For any sub-branch, the offloading decision of the previous node of the block decides the transmission energy consumption of the first task node in SB. If $e\langle T_i, T_j\rangle$ is an over-block edge, we need to consider two different situations. If T_i is the previous node of the block and it will be offloaded, then when the first task of the SB is offloaded, the transmission energy consumption is not calculated as they are both executed on the edge. Otherwise, the transmission energy consumption must be calculated when the first task of the SB is offloaded. In order to solve this problem, when T_i is not the previous node of the block, a special non-offloaded virtual node $T_s^{'}$ is added between the start node T_s and the first task node T_1, and set $x_s^{'} = y_s^{'} = z_s^{'} = 0$. In this way, we account the energy consumption for transmission of the first node of the SB. As shown in Fig. 8, the nodes on the branch are constructing a linear structure. Then, we use the linear strategy to calculation the energy consumption of SB [22]. Otherwise, if the remaining tasks within the block still compose a complex WDG, the Graph4Edge-Nonlinear strategy must be recursively called to search for the best offloading decision plan S_{min}.

For an out-block edge $e\langle T_i, T_j\rangle$, the previous task of sub-branch is unknown. For example, for calculating the out-block edge weight $e\langle T_3, T_8\rangle$ in Fig. 4, the optimal offloading decision S_{min} is necessary for the sub-branch $\{T_5, T_6\}$. However, S_{min} depends on the status of T_1 and T_2. Therefore, multiple ETGs for WDG must be constructed to measure the out-block edge's weight. Specifically, the minimum energy consumption strategy is the minimum length path among all ETGs. The specific steps for Graph4Edge-Nonlinear strategy are shown below:

Step 1: Construct the initial ETG of WDG. An arbitrary branch in WDG is chosen as the main branch. At the same time, the energy edges are added to construct ETG. And for the set of $\{T_i \mid T_i \in localSet\}$, the edges are pruned when T_i serves as the head or tail.

Step 2: Set the weight of edges in the ETG. The weights of the ordinary and in-block edges are set by Eq. (7). For the over-block edges, the Graph4Edge-Nonlinear strategy is recursively called to find its S_{min}, then set the weights by Eq. (8). Finally, the weight of all out-block edges is set to infinity. The initial ETG is shown in Fig. 6.

Step 3: Construct two different branch ETG models based on in-block edge situations. The specific description is as follows:

(1) *If the in-block edge is not from the previous node of the block, and firstly discovered.* A new ETG is created, and then the Graph4Edge-Nonlinear strategy processes the sub-branch in the block to find the optimal energy consumption offloading decision. For example, when we find $e\langle T_u, T_a\rangle$ in Fig. 6, mark the current situation and create a new ETG to record according to the current ETG. First of all, the information of the current ETG is copied to the new ETG. Then, we prune all the in-block edges which head from the previous task of block, which ensures the correct calculation of the sub-branch's minimum energy consumption strategy. At this time, the ETG generated by the linear branch WDG is shown in Fig. 8a. Finally, the weights of all out-block edges for this block in ETGs are updated.

(2) *If the in-block edge is from the adjacent predecessor task of block, and this situation was first discovered.* We can make adjustments in the current ETG. Specifically, when the in-block edges from the previous task node of this block were discovered for the first time, such as $e\langle T_v, T_b\rangle$ in Fig. 7, situation (1) has been completely traversed, so it can be processed directly on the current ETG. Prune all in-block edges that are not from the previous task node, and it ensures that

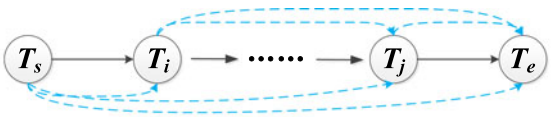
Strategy: Graph4Edge-Nonlinear

Input: A workflow dependency graph (WDG); local-execution tasks in WDG ($localSet$);
The workflow task's deadline constraint ($Deadline$);
Output: tasks in WDG (S);

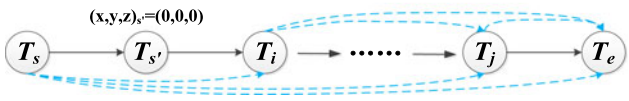
Compute (x_i, y_i, z_i) for all tasks by Eqs. (1), (2), (3);
Add T_s, T_e into WDG and set attributes;
if T'_s is needed **then**
  Add a virtual task T'_s succeed T_s and set attributes;
end if
if WDG is linear workflow **then**
  return Graph4Edge-Linear (WDG, $localSet$, $Deadline$);
end if
Get a main branch MB from WDG and construct ETG;
Prune $e(T_i, T_j)$ if T_i or T_j in $localSet$;
Set all out-block edges $e\ (T_i, T_j) = \infty$;
Compute weight for other edges by Eq. (7-8);
$ETG\_Set = ETG_{init}$;
for each in-block edge $e(T_i, T_j)$ in ETG_Set **do**
  if $isFirstFind(T_i)$ ***and*** $notPreviousNode(T_i)$ **then**
    $ETG_{Set} \leftarrow ETG$;
    $S_{SB} \leftarrow Iterate$ Graph4Edge-Nonlinear;
  end if
  if $isPreviousNode(T_i)$ **then**
    $S_{SB} \leftarrow Iterate$ Graph4Edge-Nonlinear (need T'_s);
  end if
  Compute out-block edge in ETG_Set by Eq. (8);
end for
for $k = n+1$ down to 1 **do**
  $P_{min} = Dijkstra\_Algorithm(T_s, T_k, ETG)$;
  $S = P_{min}\ (T_s, T_k)$ traversed tasks;
  if $Time_{cost} < Deadline$ **then**
    break;
  end if
end for
if $k = 0$ **then**
  $S = null$;
end if
return S;

all sub-branches can be directly calculated using the Graph4Edge-Nonlinear strategy. The ETG created by the linear branch WDG is shown in Fig. 8b. Finally, all out-block edge weights are updated for this block in ETGs.

Step 4: Use the Dijkstra algorithm to search the minimum length path in ETGs, and perform verification to ensure that deadline constraints are met. The nodes on the shortest path



(a) In-block edges not from the previous node



(b) In-block edges from the previous node

Fig. 8. Construct the ETG for branch structure.

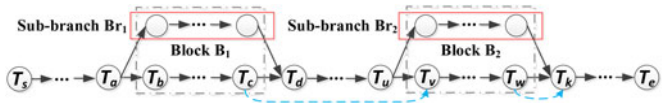


Fig. 9. WDG with multiple serial blocks.

are the minimum energy consumption offloading strategy have found.

4.2.2 Multiple-Blocks WDG

In real workflow-based applications, WDG's structures can be complex with multiple blocks in the WDG. Therefore, Graph4Edge-Nonlinear strategy should be able to deal with multiple-blocks in the WDG.

A WDG may consist of many blocks. At first, any branch can be selected as the main branch. This main branch is used to construct the initial ETG. Then, multiple ETGs are built for different blocks. In the calculation process of out-block and over-block edge weights, two new situations need to search minimum energy consumption offloading decision for the sub-branch.

(1) *WDG with multiple serial blocks.* In this situation, there is an edge that is both an out-block edge for one block and an in-block edge for another block, e.g., $e\langle T_c, T_v\rangle$ in Fig. 9. In our strategy, depending on the head node of the in-block edge for B1, the offloaded strategy with the sub-branch is different. As a result, both the head node and the in-block edge weight for Br_2 will change. In order to calculate the out-block edge weight for B2, e.g., $e\langle T_w, T_k\rangle$, the offloading strategy of Br_2 for B2 must make sure, which depends on the offloading strategy of SB1 for B1. So it is necessary to find its minimum energy consumption offloading strategy from Br_1 of B1.

(2) *WDG with nested branches.* In this situation, it is necessary to recursively call the Graph4Edge-Nonlinear strategy to find its optimal offloading strategy. For example, $e\langle T_a, T_c\rangle$ in Fig. 10 is an in-block edge of blocks B1 and B2, multiple new ETGs should be created based on the different situations of the two blocks, to find the optimal offloading strategy of sub-branches Br_1 and Br_2. Hence it is necessary to recursively call the Graph4Edge-Nonlinear strategy for the WDG $Br_1 \cup Br_2$.

The ETG for an example complex WDG is shown in Fig. 11. By recursively calling the Graph4Edge-Nonlinear strategy for the sub-branches, the minimum energy consumption offloading decision of the whole WDG can be found. For example, given an in-block edge $e\langle T_i, T_j\rangle$ in Fig. 11, the Graph4Edge-Nonlinear strategy calculates the sub-branch $\{T_u | T_u \in WDG \wedge T_u \rightarrow T_k \wedge T_u \nrightarrow T_j \wedge T_u \nrightarrow T_h\}$, and gets the weight of out-block $e\langle T_h, T_k\rangle$.

4.2.3 Strategy Description

Clearly, no matter how complicated the structure of the WDG is, it can always be transformed to the linear structure by calling Graph4Edge-Nonlinear strategy recursively.

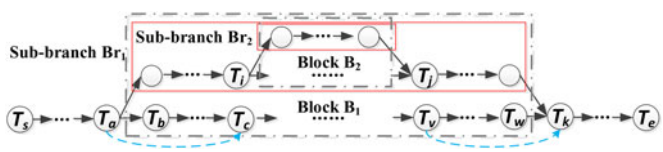


Fig. 10. WDG with nested branches.

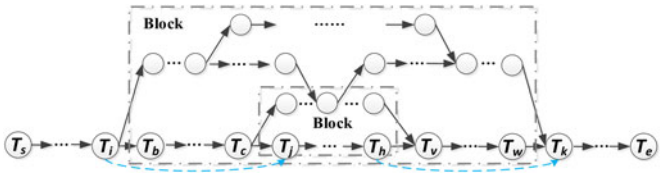


Fig. 11. The ETG of WDG.

Here, we present the pseudo-code for the Graph4Edge-Nonlinear strategy.

First, WDG is initialized (Lines 1-5). If the WDG is a linear structure, the Graph4Edge-Linear strategy is directly called (Line 7). Otherwise, an arbitrary branch from T_s to T_e is chosen as the main branch, and this main branch is used to construct the initial ETG (Lines 9-10), and compute the weight of the ordinary, in-block and over-block edge by Eq. (7-8) (Line 12). Next, all the in-block edges are traversed in sequence. When the in-block edge of a block is found for the first time, and its head node is not the previous task for this block, a new ETG is created and added to the *ETG_Set*(Lines 15-18). Then, the shortest path of the sub-branch can be found in the new ETG. When an in-block edge of which the head node is the previous task of this block is found, the current ETG is processed (Lines 19-21) to obtain the sub-branch's optimal offloaded strategy. Meanwhile, the weight of the out-block edges is updated in *ETG_Set*(Line 22). Finally, the Dijkstra algorithm can find the shortest path from T_s to T_k (Line 25). In the worst situation (Line 31), if all solutions do not meet the deadline constraints, all tasks in the workflow are executed locally.

5 Preformance Evalution

In this section, we first revisit our motivating example as a real-world case study to illustrate the effectiveness of our strategy. Afterwards, we describe the simulation environment and parameter settings, followed by comprehensive simulation experiments. We evaluate the performance of Graph4Edge-Nonlinear and compare with other representative computation offloading strategies in the end-device's energy consumption, strategy execution time and task response time based on both the real-world data of UAV-EXPRESS and the simulated workflows using the FogWorkflowSim platform. UAV-EXPRESS is a prototype UAV delivery system developed based on EXPRESS which is an energy-efficient and secure framework for MEC and blockchain [23]. FogWorkflowSim is a simulation platform for workflow applications with the focus on the performance evaluation of computation offloading and task scheduling strategies [25].

5.1 Case Study

Similar to the motivating example shown in Fig. 1, Fig. 12 shows the detailed workflow for the final parcel delivery process in the UAV delivery system [24]. In our delivery system, the UAV and the edge server is connected by 100 Mbps WiFi. Before the UAV reaches the destination for parcel delivery, the edge server downloads the facial images of the parcel receiver from the cloud server of the logistics system. When the UAV arrives at the destination, the video stream captured by the camera of the UAV is analyzed

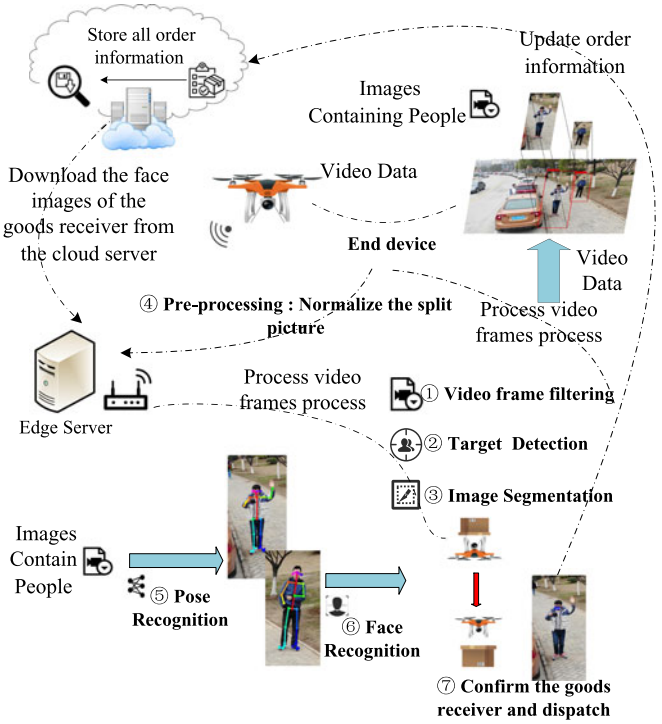


Fig. 12. A case study on the MEC-Based UAV delivery system.

frame by frame to locate the position and confirm the identity of the parcel receiver.

The final parcel delivery process can be described in three stages. In the first stage (including Tasks 1-3), the original video frames are filtered on the UAV. We use the target detection function to search for video frames containing people, and frames without people are filtered directly. A small proportion of the images containing people in the whole video frame is further extracted by using the image segmentation algorithm according to the detected location coordinate of the people. In the second stage (including Task 4 and Task 5), extracted video frames are further processed. In a real-world scenario, there could be a lot of pedestrians near the destination, and hence there will be many images containing multiple people. We use pose recognition (i.e., the parcel recipient will receive an instruction on her/his mobile App to make a specific pose such as waving the right hand from right to left) to further identify the receiver from the crowd. Finally, in the third stage (including Task 6 and Task 7), identification of the parcel receiver is conducted using face recognition. Face recognition confirms whether the person waving hand matches the face images downloaded in advance from the cloud server. If matches, the UAV will approach the receiver for landing and then hand over the goods. Otherwise, more image frames are processed until the correct parcel receiver is found, or the delivery process terminates without successfully locating the parcel receiver.

For the experiment, we select a one-minute video clip, with 1920*1080 pixel and 30 FPS recorded by the DJ Mavic Air UAV[1]. With our computation offloading strategy, the generated computation offloading decision is that Task 1-3

1. https://www.dji.com/cn/mavic-air

TABLE 1
MEC Environment Parameter Setting

Parameters	End Server	Edge Server
MIPS	1000	1300
Load Power (mW)	700	N/A
Idle Power (mW)	30	N/A
Data Transmission Power (mW)	100	N/A

TABLE 2
PSO, GA Strategy Parameter Setting

PSO		GA	
Parameters	Setting	Parameters	Setting
Particles	30	Particles	50
Iterations	100	Iterations	100
Factor C1, C2	2	Cross Rate	0.8
Inertia Weight	1	Mutation Rate	0.1
Repeated	10	Repeated	10

and Task 4 are to be executed locally, while Task 5 and Task 6 are to be offloaded to the edge server. Specifically, Task 1-3 and Task 4 are the video pre-processing tasks that can be executed on the UAV since the characteristic of data size but the required computing power is low. After the UAV completes the video pre-processing tasks, the remaining data size is reduced to 10% of the original video frames. The deep neural network-based computation tasks (Task 5 and Task 6) are executed at the edge server because the size of data transfer for Task 5-6 is very small but they both require high computing power.

5.2 Simulation Environment and Parameter Settings

In our experiments, we compare Graph4Edge-Nonlinear with three types of computation offloading strategies. According to the pseudo-code of Graph4Edge-Nonlinear given above, the worst time complexity is $O(n^3)$ and the space complexity is about $O(n^3)$, does not exceed $O(n^4)$. For the comparsion algorithms, the first type is based on search algorithms including PSO and GA which are most widely used for computation offloading. According to [32], the average time complexity of the standard PSO algorithm is $O(n^3)$, and the worst is $O(n!)$. The time complexity of the GA algorithm [33] is about $O(n^2)$, but multiple iterations will bring additional time overhead. The second type is a Greedy strategy which makes the offloading decision for each task through comparison the energy consumption required for offloading with local execution. Obviously, Greedy only needs one traversal, the time complexity is $O(n)$. If the energy consumption of task offloading is less than execution locally, then the task will be offloaded to the edge server for execution. Otherwise, the task will be executed locally. The third type is the All-in-End strategy, which means that all of the tasks are executed in the end device. Although the time complexity of the All-in-End strategy is only $O(1)$, the extremely high energy consumption will seriously affect the quality of service. All computation offloading strategies are applied with the Min-Min task scheduling algorithm [31] at the edge server.

All simulation experiments are implemented on the Fog-WorkflowSim platform, which is a simulation platform for Fog/MEC-based workflow applications [25]. It supports different kinds of workflow structures and different evaluation index metrics such as time, energy and cost. The experiments are run on a laptop with the following configuration: Intel Core™ I7-9750H CPU 2.60GHz, 16G RAM, NVIDIA GeForce GTX 1660Ti.

Table 1 describes experimental parameter settings of the MEC environment. The simulated MEC environment consists of three edge servers and one end-device. The processing speed of the computing resources is randomly chosen between 100 and 1500 Megacycles. According to the EXRESS framework and actual data collection in UAV last-mile delivery scenarios, the input and output data size for each task is generated between 0.625 and 30 MB randomly [23], [32], [33]. In this paper, we only consider the energy consumption of the end-device. In order to fairly evaluate our strategy and other strategies, we define the data transmission rate between end-device and edge server as 100 Mbps and the bandwidth is assumed to be relatively stable [32], [34].

Table 2 describes the parameter settings of PSO and GA strategy respectively [35], [36]. For each workflow application, we simulate 100 times to obtain the average result. In order to comprehensively evaluate the overall performance, we randomly generate many complex WDG of different sizes from 10 to 100 tasks with both linear and nonlinear workflow structures. The percentage of the local execution task (namely those cannot be offloaded to the edge) is set as 20%.

5.3 Simulation Evaluation

Now we present the detailed simulation results. Deadline constraint is the most important QoS constraint in any real-world business systems. According to actual business requirements, there is usually a strict time constraint. Based on the results of our actual program [23], our workflow application's deadline constraint is set between 70% and 140% of the total task local execution time. Note that the deadline constraint considered in this paper is the soft deadline, which means that missing the deadline will not cause task failures but only decrease the service quality.

Fig. 13 shows the energy consumption results under different deadline constraints. Initially, all tasks are executed

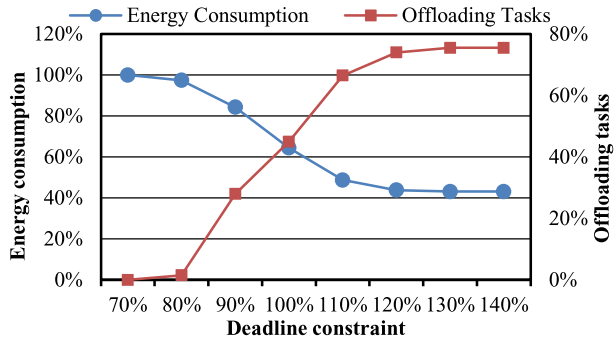


Fig. 13. Energy consumption and task's offloading per-cent with different deadline constraints.

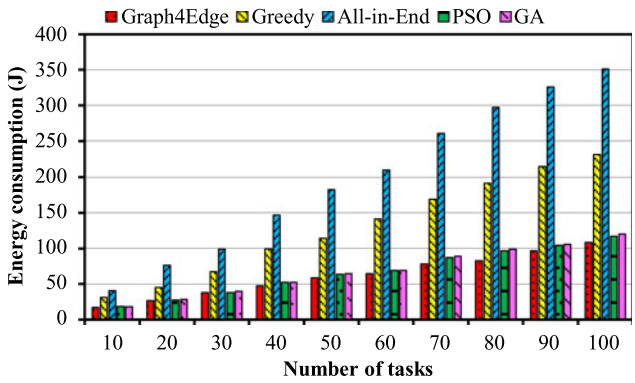


Fig. 14. Comparison of Graph4Edge-Nonlinear and other four methods for energy consumption.

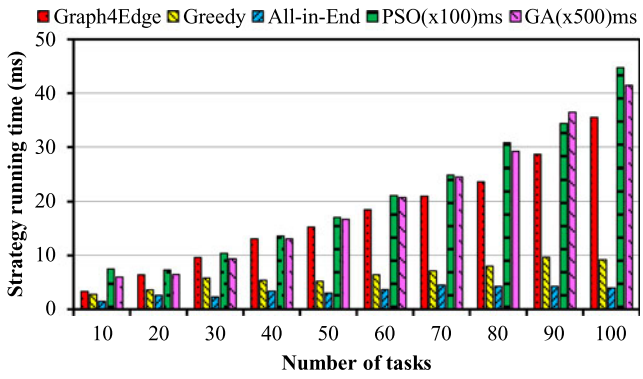


Fig. 15. Comparison of Graph4Edge-Nonlinear and other four methods for strategy running time.

locally on the end-device. Initially, all tasks are executed locally on the end-device. When the value of deadline constraint increases, the percentage of offloaded tasks also increases. It can be seen that the end-device's energy consumption becomes stable when the deadline constraint reaches 130%. The results show that when the deadline constraint becomes more flexible, the room for the computation offloading strategy becomes larger and hence our Graph4Edge-Nonlinear strategy becomes more effective in reducing the end-device's energy consumption. However, when deadline constraint reaches a certain value, the effectiveness of reducing energy consumption levels off since the percentage of offloaded tasks also reaches its maximum 80% (as we have set 20% of local execution tasks). In order to compare the best performance of different computation offloading strategies, we focus on the deadline is 130% (30% extra time than execute locally) in the following experiments.

To comprehensively evaluate its performance, we compare Graph4Edge-Nonlinear with others in energy consumption, strategy running time and task response time.

Fig. 14 shows that the end-device's energy consumption of the over different sizes of nonlinear workflows. The experimental result demonstrates that energy consumption with Graph4Edge-Nonlinear is always lower than the other four strategies. For example, when the task number is 50, the energy consumption with Graph4Edge-Nonlinear is 7.81% and 9.51% lower than PSO and GA respectively. The Greedy strategy and All-in-End strategy always have higher energy consumption than Graph4Edge-Nonlinear which is about 94% and 310% respectively. This is because they place too many tasks on the end device for execution.

Fig. 15 illustrates the results of strategy running time over different sizes of nonlinear workflows. Please be noted that the basic time units for Graph4Edge-Nonlinear, Greedy and the All-in-End strategies are milliseconds (ms), while the basic time units for PSO and GA are 100ms and 500ms respectively. Clearly, Graph4Edge-Nonlinear is much faster than search-based strategies PSO and GA. Specifically, it is running about 110 times and 540 times faster than PSO and GA respectively. In the FogWorkflowSim platform [25], All-in-End strategy running needs to count the number of tasks that make up the workflow, which usually takes a few milliseconds. Although the Greedy and All-in-End strategies have the smallest execution time, it always has the worst performance in reducing energy consumption. In particular, even if the number of tasks is as high as 100, the Graph4Edge strategy only needs an additional 30ms to get the optimal offloading decision. When it is necessary to perform offloading of delay-sensitive tasks, our strategy can guarantee real-time performance.

It is also important to investigate the impact of workflow structures on the strategy running time. Fig. 16 compares the strategy running time of Graph4Edge-Nonlinear and linear strategy on nonlinear workflows and linear workflows respectively with 10 to 50 tasks. The result shows that with the same workflow sizes, the strategy running time of Graph4Edge-Nonlinear is about 20% higher than linear strategy, which is not a significant increase considering the much more complex structures of the nonlinear workflows. Meanwhile, even with 50 tasks, the strategy running time of Graph4Edge-Nonlinear is only increased by 2ms.

Fig. 17 illustrates the results of task response time over different sizes of tasks in nonlinear workflows. The workflow tasks are executed according to the offloading decision plan. In FogWorkFlowSim platform [25], the task response time is defined as the sum of task execution time and data transfer time. Note that the strategy execution time is not included since it is trivial compared with task execution time. Obviously, our proposed Graph4Edge-Nonlinear strategy requires minimal task response time.

In summary, given the experimental results above, we can conclude that the Graph4Edge-Nonlinear strategy is able to find the optimal computation offloading decision with the lowest energy consumption under the given deadline constraint for a complex workflow application. Most importantly, for delay-sensitive applications, our proposed

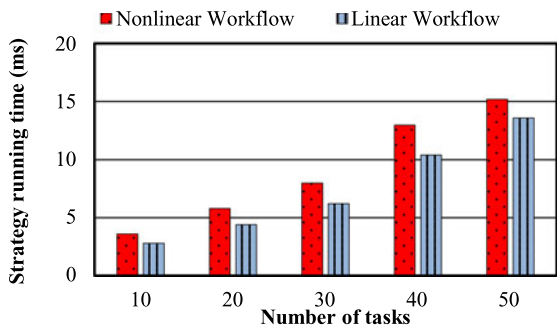


Fig. 16. Comparison of nonlinear workflows and linear workflows for strategy running time.

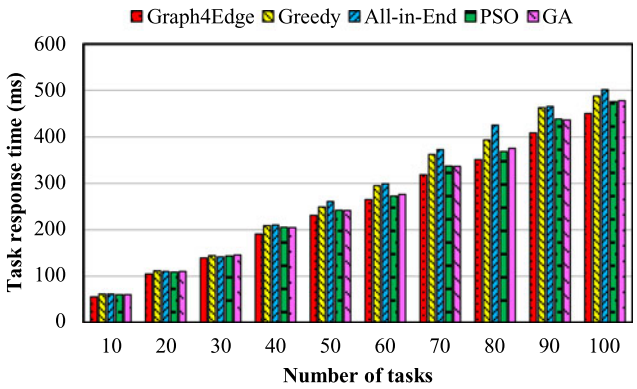


Fig. 17. Comparison of Graph4Edge-Nonlinear and other four methods for task response time.

strategy can meet their real-time requirements given its faster running time.

6 Related Work

In a MEC environment, the main objective of the computation offloading strategy is to find the best offloading decision according to the characteristics of tasks, computing resources and network conditions [37]. In addition, these methods can improve the utilization rate of computing resources in MEC [38] and minimize the end-device's energy consumption with the QoS constraints of users and reduce the cost of the service providers [39]. Computation offloading is used for solving the problems caused by insufficient computing power and insufficient battery capacity of the end-device. With the gradual promotion of MEC platform, computation offloading has become an important research topic [28]. There are many preliminary research works focusing on the problem of computation offloading in the MEC environment.

Currently, a range of research works focuses on the problem of computation offloading to reduce end-device's energy consumption and task response time. In the aspect of the energy consumption optimization, Zhang *et al.* [40] focused on the multi-access characteristics of the 5G and an minimize the energy consumption offloading strategy in 5G MEC networks is proposed. The [41] proposed online multi-task offloading schemes for MEC system. For the target of time optimization, Xing *et al.* [42] propose a MEC system that can reduce the task's computation latency significantly. Currently, many research works are based on popular optimization algorithms such as PSO and GA [35], [36]. However, most studies only focused on the independent tasks, and ignored the computational overhead needed to make decisions. Most of the tasks in real-world applications are highly correlated. Each task depends on the execution result of the previous task and provides the necessary data flow for the successor task [19], [32].

Graph as a kind of classical data structure. Many research works about graph-based algorithms in Cloud and Edge Computing. In the cloud computing environment, many studies adopted graph-based algorithms to solve task management problems for different workflow structures. Yuan *et al.* [43] aimed at the problem of data set storage in data-intensive scientific workflow execution and proposed the CTT-SP algorithm to trade-off computation and storage cost in the cloud environment. In recent years, there are also many studies in the MEC environment. Khare *et al.* [44] focused on the data placement problem of operators in streaming applications and proposed an algorithm to convert streaming DAG into a set of approximate linear chains and perform data placement and time prediction. Zhang *et al.* [45] proposed an efficient and large-scale graph computing adaptive solution named GraphA, which can achieve fine-grained and low-cost graph structure data storage. Most research focuses on resource management issues in cloud or edge conditions. However, these works have not paid attention to the problems of end-device's limited computing power and battery capacity in the MEC environment. In [22], the authors proposed a graph-based strategy that can provide a solution for the generation of the optimal energy consumption offloading strategy for linear workflow, but this strategy cannot handle the nonlinear workflow of complex applications.

Most existing works focused on reduce end-device's energy consumption and task response time, but very few studies take into account the complexity of scientific workflow tasks in real-world. However, in the MEC-based UAV delivery system, the existing tasks with nonlinear structure can seriously affect the QoS. Due to the complexity of computing and delivery services in the MEC environment and the requirement of real-time response in various smart system scenarios, traditional cloud-based computing offloading strategies and heuristic search algorithms cannot be effectively utilized. Therefore, this paper deals with various problems of computation offloading in the MEC environment using graph-based technology. We focus on workflow applications with complex structures (viz. parallel, selective, and iterative) and propose the Graph4Edge-Nonlinear strategy as an effective solution.

7 Conclusion and Future Work

Computation offloading is a key technology to optimize the QoS of MEC-based applications. However, most existing strategies did not pay more attention to the dependency between computing tasks or are usually based on heuristic search algorithms. This is unacceptable for delay-sensitive applications. For solving these problems, a novel graph-based computation offloading strategy to minimize the end-device's energy consumption under the given deadline constraint is proposed in this paper. Motivated by a MEC-based UAV delivery system, we first built the nonlinear workflow model for complex applications. Then, using the graph-based partition technique, we proposed the Graph4Edge-Nonlinear strategy to search for the best computation offloading decision with the lowest energy consumption under the deadline constraint. Finally, both a real-world case study and comprehensive simulation experiments implemented on the FogWorkflowSim platform with different workflow structures and data sizes are conducted to evaluate the effectiveness of our proposed strategy. The experimental results have shown that Graph4Edge-Nonlinear can achieve overall better performance with strategy running time and energy consumption than other representative computation offloading strategies.

This paper mainly focused on reducing end-devices energy consumption and improving decision-making efficiency in the MEC environment. In the future, we will investigate the problem of computation offloading together with workflow scheduling at the edge servers, and take the energy consumption of edge servers into account to produce a holistic solution that can improve the QoS for the whole MEC-based delivery system. Besides, the dynamic fluctuation of the network in the real world and the signal loss caused by end-device moving will also be further considered.

REFERENCES

[1] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog A survey and analysis of security threats and challenges," *Future Gener. Comput. Syst.*, vol. 78, no. 1, pp. 680–698, 2018.

[2] S. Kekki *et al.*, "MEC in 5G networks," *ETSI White Paper*, vol. 28, no. 1, pp. 1–28, 2018.

[3] S. K. Battula, S. Garg, J. Montgomery, and B. Kang, "An efficient resource monitoring service for fog computing environments," *IEEE Trans. Serv. Comput.*, vol. 13, no. 4, pp. 709–722, Jul./Aug. 2020.

[4] X. Lyu *et al.*, "Selective offloading in mobile edge computing for the green Internet of Things," *IEEE Netw.*, vol. 32, no. 1, pp. 54–60, Jan./Feb. 2018.

[5] Y. Jiang, "A survey of task allocation and load balancing in distributed systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 585–599, Feb. 2016.

[6] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture," in *Proc. IEEE Symp. Comput. Commun.*, 2012, pp. 1–8.

[7] Z. Kuang, L. Li, J. Gao, L. Zhao, and A. Liu, "Partial offloading scheduling and power allocation for mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6774–6785, Aug. 2019.

[8] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros, "Workflow patterns," *Distrib. Parallel Databases*, vol. 14, no. 1, pp. 5–51, 2003.

[9] X. Fei, S. Lu, and C. Lin, "A mapreduce-enabled scientific workflow composition framework," in *Proc. IEEE Int. Conf. Web Serv.*, 2009, pp. 663–670.

[10] F. Zhou, Y. Wu, H. Sun, and Z. Chu, "UAV-enabled mobile edge computing: Offloading optimization and trajectory design," in *Proc. IEEE Int. Conf. Commun.*, 2018, pp. 1–6.

[11] A. Alioua, H. E. Djeghri, M. E. T. Cherif, S. M. Senouci, and H. Sedjelmaci, "UAVs for traffic monitoring: A sequential game-based computation offloading/sharing approach," *Comput. Netw.*, vol. 177, no. 1, pp. 1–15, 2020.

[12] W. Song, L. Wang, R. Ranjan, J. Kolodziej, and D. Chen, "Towards modeling large-scale data flows in a multidatacenter computing system with petri net," *IEEE Syst. J.*, vol. 9, no. 2, pp. 416–526, Jun. 2015.

[13] H. Cao, H. Jin, S. Wu, and S. Ibrahim, "Petri net based grid workflow verification and optimization," *J. SuperComputing*, vol. 66, no. 3, pp. 1215–1230, 2013.

[14] Z. Guan *et al.*, "Grid-Flow: A grid-enabled scientific workflow system with a petri-net-based interface," *Concurrency Computation: Pract. Experience*, vol. 18, no. 10, pp. 1115–1140, 2006.

[15] W. Van Der Aalst, K. M. Van Hee, and K. van Hee, *Workflow Management: Models, Methods, and Systems*. Cambridge, MA, USA: MIT Press, 2004.

[16] J. Meng, H. Tan, X. Li, Z. Han, and B. Li, "Online deadline-aware task dispatching and scheduling in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1270–1286, Jun. 2020.

[17] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. M. Leung, "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing," *IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2651–2664, Dec. 2018.

[18] X. Xu *et al.*, "An energy-aware computation offloading method for smart edge computing in wireless metropolitan area networks," *J. Netw. Comput. Appl.*, vol. 133, no. 1, pp. 75–85, 2019.

[19] S. Deng, L. HUang, J. Taheri, and A. Y. Zomaya, "Computation offloading for service workflow in mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3317–3329, Dec. 2015.

[20] L. Yu, J. Ge, S. Zhang, J. Wu, Z. Tang, and B. Luo, "A utility-based optimization gramework for edge service entity caching," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 11, pp. 2384–2395, Nov. 2019.

[21] M. Hu, L. Zhuang, D. Wu, Y. Zhou, X. Chen, and L. Xiao, "Learning driven computation offloading for asymmetrically informed edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 8, pp. 1802–1815, Aug. 2019.

[22] L. Fan, X. Liu, X. Li, D. Yuan, and J. Xu, "Graph4Edge: A graph-based computation offloading strategy for mobile-Edge workflow applications," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops*, 2020, pp. 1–4.

[23] J. Xu, X. Liu, X. Li, and Y. Yang, "Express: An energy-efficient and secure framework for mobile edge computing and blockchain based smart systems," in *Proc. 35th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2020, pp. 1283–1286.

[24] H. Gao *et al.*, "Edge4Sys: A device-edge collaborative framework for MEC based smart systems," in *Proc. 35th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2020, pp. 1252–1254.

[25] X. Liu *et al.*, "FogWorkflowSim: An automated simulation toolkit for workflow performance evaluation in fog computing," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2019, pp. 1114–1117.

[26] G. K. Garge and C. Balakrishna, "Unmanned aerial vehicles (UAVs) as on-demand QoS enabler for multimedia applications in smart cities," in *Proc. Int. Conf. Innov. Intell. Inform. Comput. Technol.*, 2018, pp. 1–7.

[27] N. H. Motlagh, M. Bagaa, and T. Taleb, "UAV-based IoT platform: A crowd surveillance use case," *IEEE Commun. Mag.*, vol. 55, no. 2, pp. 128–134, Feb. 2017.

[28] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surv. Tut.*, vol. 17, no. 3, pp. 1628–1656, Jul.–Sep. 2017.

[29] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang, "A market-oriented hierarchical scheduling strategy in cloud workflow systems," *J. Supercomputing*, vol. 63, no. 1, pp. 256–293, 2013.

[30] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A survey of data-intensive scientific workflow management," *J. Grid Comput.*, vol. 13, no. 4, pp. 457–493, 2015.

[31] K. Etminani and M. Naghibzadeh, "A min-min max-min selective algorihtm for grid task scheduling," in *Proc. 3rd IEEE/IFIP Int. Conf. Central Asia Internet*, 2007, pp. 1–7.

[32] J. Xu *et al.*, "Mobility-aware workflow offloading and scheduling strategy for mobile edge computing," in *Proc. Int. Conf. Algorithms Architectures for Parallel Process.*, 2019, pp. 184–199.

[33] T. Zhu, T. Shi, J. Li, Z. Cai, and X. Zhou, "Task scheduling in deadline-aware mobile edge computing systems," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4854–4866, Jun. 2019.

[34] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in Fog-cloud computing toward balanced delay and power consumption," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1171–1181, Dec. 2016.

[35] A. Kaswan, V. Singh, and P. K. Jana, "A multi-objective and PSO based energy efficient path design for mobile sink in wireless sensor networks," *Pervasive Mobile Comput.*, vol. 46, no. 1, pp. 122–136, 2018.

[36] H. Hallawi, J. Mehnen, and H. He, "Multi-capacity combinatorial ordering GA in application to cloud resources allocation and efficient virtual machines consolidation," *Future Gener. Comput. Syst.*, vol. 69, no. 1, pp. 1–10, 2017.

[37] J. Hu, M. Jiang, Q. Zhang, Q. Li, and J. Qin, "Joint optimization of UAV position, time slot allocation, and computation task partition in multiuser aerial mobile-edge computing systems," *IEEE Trans. Veh. Technol.*, vol. 68, no. 7, pp. 7231–7235, Jul. 2019.

[38] M. G. R. Alam, M. M. Hassan, M. Z. Uddin, A. Almogren, and G. Fortino, "Autonomic computation offloading in mobile edge for IoT applications," *Future Gener. Comput. Syst.*, vol. 90, no. 1, pp. 149–157, 2019.

[39] S. Wang *et al.*, "QoS prediction for service recommendations in mobile edge computing," *J. Parallel Distrib. Comput.*, vol. 127, no. 1, pp. 134–144, 2019.

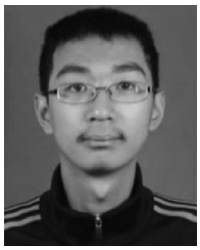
[40] S. Wang, Y. Zhao, L. Huang, J. Xu, and C. H. Hsu, "Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks," *IEEE Access*, vol. 4, no. 1, pp. 5896–5907, 2016.

[41] W. Chen, D. Wang, and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Trans. Serv. Comput.*, vol. 12, no. 5, pp. 726–738, Sep./Oct. 2019.
[42] H. Xing, L. Liu, J. Xu, and A. Nallanathan, "Joint task assignment and resource allocation for D2D-enabled mobile-edge computing," *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4193–4207, Jun. 2019.
[43] Y. Dong, L. Cui, W. Li, X. Liu, and Y. Yang, "An algorithm for finding the minimum cost of storing and regenerating datasets in multiple clouds," *IEEE Trans. Cloud Comput.*, vol. 6, no. 2, pp. 519–531, Apr.–Jun. 2018.
[44] S. Khare *et al.*, "Linearize, predict and place: Minimizing the makespan for edge-based streamprocessing of directed acyclic graphs," in *Proc. ACM/IEEE Symp. Edge Comput.*, 2019, pp. 1–14.
[45] Y. Zhang, D. Li, C. Zhang, J. Wang, and L. Liu, "GraphA: Efficient partitioning and storage for distributed graph computation," *IEEE Trans. Serv. Comput.*, vol. 14, no. 1, pp. 155–166, Jan. 2021.

Xuejun Li (Member, IEEE) received the PhD degree in computer application technology from the School of Computer Science and Technology, Anhui University, Hefei, Anhui, China, in 2008. He is currently a full professor with the School of Computer Science and Technology, Anhui University, Hefei, Anhui, China. His major research interests include mobile edge computing, workflow systems, cloud computing, and intelligent software.

Tianxiang Chen received the bachelor's degree in Internet of Things engineering from the School of Computer and Information Engineering, Fuyang Normal University, Fuyang, Anhui, China, in 2018. He is currently working toward the master's degree with the School of Computer Science and Technology, Anhui University, Hefei, Anhui, China. His current research interests include mobile edge computing, workflow system, deep learning, resource management.

Dong Yuan (Member, IEEE) received the BEng and MEng degrees in computer science from Shandong University, Jinan, China, in 2005 and 2008, respectively, and the PhD degree in computer science from the Swinburne University of Technology, Melbourne, Australia, in 2012. He is a senior lecturer with the School of Electrical and Information Engineering, University of Sydney, Sydney, Australia. His research interests include cloud computing, parallel and distributed systems, scheduling and resource management, deep learning, data management and Internet of Things.



Jia Xu (Member, IEEE) received the bachelor's and master's degrees in computer science and technology from the School of Computer Science and Technology, Anhui University, Hefei, Anhui, China, in 2010-2017, respectively. He is currently working toward the PhD degree with the School of Computer Science and Technology, Anhui University, Hefei, Anhui, China. He was a software engineer focusing on industrial projects and solutions in iFLYTEK Company, Ltd from 2017-2018. His current research interests include mobile edge computing, workflow system, cloud computing, resource management.

Xiao Liu (Senior Member, IEEE) received the bachelor's and master's degrees in information management and information system from the School of Management, Hefei University of Technology, Hefei, China, in 2004 and 2007, respectively, and the PhD degree in computer science and software engineering from the Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne, Australia, in 2011. He was an associate professor with the Software Engineering Institute, East China Normal University, Shanghai, China. He is currently an associate professor with the School of Information Technology, Deakin University, Melbourne. His current research interests include software engineering, distributed computing, and data mining, with special interests in workflow systems, cloud/fog computing, and human-centric software engineering.