Hindawi
Complexity
Volume 2020, Article ID 9548262, 14 pages
https://doi.org/10.1155/2020/9548262

Research Article

Energy- and Resource-Aware Computation Offloading for Complex Tasks in Edge Environment

Kai Peng,[1] **Bohai Zhao**,[1] **Shengjun Xue**,[2] **and Qingjia Huang**[3]

[1]*College of Engineering, Huaqiao University, Quanzhou, China*
[2]*School of Computer Science and Technology, Silicon Lake College, Suzhou, China*
[3]*Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China*

Correspondence should be addressed to Shengjun Xue; sjxue@163.com

Received 7 November 2019; Revised 17 December 2019; Accepted 6 January 2020; Published 26 March 2020

Academic Editor: Hassan Zargarzadeh

Mobile users typically have a series of complex tasks consisting of time-constrained workflows and concurrent workflows that need to be processed. However, these tasks cannot be performed directly locally due to resource limitations of the mobile terminal, especially for battery life. Fortunately, mobile edge computing (MEC) has been recognized as a promising technology which brings abundant resource at the edge of mobile network enabling mobile devices to overcome the resource and capacity constraints. However, edge servers, such as cloudlets, are heterogeneous and have limited resources. Thus, it is important to make an appropriate offloading strategy to maximize the utility of each cloudlet. In view of this, the time consumption and energy consumption of mobile devices and resource utilization of cloudlets have been taken into consideration in this study. Firstly, a multiconstraint workflow mode has been established, and then a multiobjective optimization mode is formulated. Technically, an improved optimization algorithm is proposed to address this mode based on Nondominated Sorting Genetic Algorithm II. Both extensive experimental evaluations and detailed theoretical analysis are conducted to show that the proposed method is effective and efficiency.

1. Introduction

With the development of Internet core technologies such as Wireless Sensor Network (WSN), Near Field Communication (NFC), and Radio Frequency Identification (RFID), the mobile devices (MDs), such as mobile phones, are in full swing in recent years [1–4]. According to Cisco VNI forecast, there will be nearly 12.3 billion mobile-connected devices by 2022; moreover, the global annual data traffic will reach almost one zettabyte [5]. In addition to the breakthrough of core technologies, the rapid development of MDs is also related to its compact, portable, and some other favorable features. However, compared with traditional devices, such as the computer, an MD still has some limitations in resources and computing capacity, which leads to mass energy and time consumption, especially for the complex tasks generated by the multiconstraint applications [6–9].

The emergence of mobile cloud computing (MCC) brings new vitality to MDs with the idea of offloading tasks to remote cloud [10–12], which has a strong pertinence in resolving the resource constraints of MDs [13–15]. Nevertheless, due to the long distance between MDs and cloud, transmission latency, which is caused by computation offloading and data transmission, is inevitable and has even been considered as the bottleneck of MCC [16–18]. Furthermore, high latency may even lead to huge energy consumption, which is intolerable for MDs [19–21], namely, computation offloading in MCC may not be able to meet our demand for the quality of service in some complicated environment.

Fortunately, a new technology named MEC has been proposed. MEC can be regarded as a special case of MCC, and its three-layer architecture is specifically designed to confront the intrinsic flaws in MCC [22–25]. And cloudlet is considered to be a new type of edge server located in the

intermediate tier of MEC to better support complex tasks [26–28]. In addition, with the development of MEC, it may realize the execution of complex tasks in an ultra-low time consumption by pushing the computing, execution, and storage to the cloudlet [29–31]. Furthermore, the inherent limitations of MDs can probably be solved, and the seamless integration of network and MDs may become promising [32–34].

However, the capacity of the cloudlet is generally limited [35–37]. If multiple tasks request for computing resources, and the number of the tasks exceeds the capacity of a cloudlet, the queue latency will be generated [38–40]. Furthermore, as the number of tasks increases, serious network congestion will occur between MDs and the cloudlet [41–43]. In addition, the applications of MD usually consist of complex tasks, namely, time-constrain workflow and concurrently workflow application (WA), if tasks are offloaded irregularly, this kind of latency may occur in each cloudlet, which may even lead to data loss or the failure of the task [44–46]. In terms of the whole network, the resource utilization of cloudlets should also be considered, which determines whether we can achieve the maximum benefit with limited cloudlet resources. Thus, it becomes very complex to propose a computation offloading strategy for complex tasks in multicloudlet environment. Moreover, the cloudlets are often heterogeneous. Hence, the properties of the cloudlet should also be taken into considered when formulating the task offloading strategy, which further increases the difficulty.

In MEC environment, some optimization strategies for WAs have been proposed (to list a few here [23, 24, 39, 41]). But how to develop a multiobjective optimization offloading strategy for complex tasks consisting of time-constrained workflows and concurrent workflows is still a big challenge, especially in the multicloudlet environment.

In view of this, in this paper, we investigated the computation offloading problem of complex tasks in multicloudlet environment. Both the energy consumption and time consumption of MDs as well as the resource utilization of cloudlets are taken into account. The main contributions of this paper can be summarized as follows:

(1) To solve the computation offloading problem for complex tasks in multicloudlet environment, we propose a multiobjective optimization algorithm for complex tasks in multicloudlet environment (MOHWE). Both energy consumption and time consumption of MDs as well as the resource utilization of cloudlets are taken into consideration.

(2) A new workflow mode is established by recursive method to evaluate complex tasks, namely, multiconstraint WAs, and then a multiobjective optimization mode is formulated. The energy consumption, time consumption of MDs, and resource utilization of cloudlets are jointly optimized while meeting the deadline constraints.

(3) Some comparative methods are proposed, such as full random offloading method, full offloading to cloudlet method, and full offloading based on the first come first service method. Extensive experiments have indicated that our method is effective and efficient with the queueing waiting time and the number of WA increases.

The rest sections of this paper are described as follows. In Section 2, the related work is summarized. In Section 3, the system mode and objective functions are proposed. Section 4 analyzes the principle of multiconstraint WAs in details, and a multiobjective optimization algorithm for the multiconstraint WAs is proposed. In Section 5, extensive experiments are carried out and the corresponding results are analyzed. Eventually, conclusion and the future work are outlined.

2. Related Work

Computation offloading is also called cyber foraging [47], the main idea is to offload the computing of tasks to cloudlet or cloud in order to liberate MDs' inherent defects of computing resources. Initially, computation offloading was studied in MCC for general applications and some critical theoretical achievements have been proposed in [6, 11, 13, 21].

Tseng et al. [6] focused on how to reduce the offloading time and execution time of applications in MCC environment. The authors proposed two optimization algorithms, realizing the multiobjective optimization of energy consumption and time consumption. Jia et al. [11] made a deep discussion on how to formulate an online offloading method to decide which task should be offloaded. They proposed a heuristic algorithm for computing intensive applications and constructed an online offloading strategy, which can minimize the execution time of tasks. Benkhelifa et al. [13] proposed a system that can schedule cloud resources intelligently. Through the coordination between the data center and the user applications, the system can make full use of resources and minimize energy consumption. Gai et al. [21] focused on the energy consumption problem caused by task offloading to remote cloud servers in cloud computing. They proposed an energy-aware management model in a heterogeneous cloud environment, and the energy consumption of mobile network applications is decreased effectively.

As the structure of the application is becoming more sophisticated, there have been many studies on WAs in MCC environment [9, 14, 18]. Technically, computing offloading for WAs is more challenging.

Deng et al. [9] focused on the computation offloading problem of service WAs and proposed a solution based on the genetic algorithm. Then, the authors constructed a compromise fault-tolerant mechanism and optimized the energy consumption and time consumption for MDs. Guo et al. [14] proposed a resource scheduling method for WAs, realizing the optimization of energy and time consumption while meeting the deadline constraints. Peng et al. [18] made a deep discussion on how to optimize the energy

consumption and time consumption of WAs in MCC environment. Through the tradeoff between system performance and energy consumption, they proposed a multiobjective optimization method for WAs based on the whale optimization algorithm.

However, due to the different architecture between MCC and MEC, the offloading strategy in MCC cannot be used in MEC. How to design a suitable offloading strategy in MEC environment is still a problem needs to be studied.

Wu et al. [12] investigated the application partition problem in mobile environment. They proposed an offloading partitioning algorithm for dynamic applications. The energy consumption, time consumption, and the cost of the applications are well optimized jointly. Xu et al. [32] focused on the information security and offloading utility problem of applications. They hold the view that serious privacy leakage will occur during the transmission process between MDs and edge servers. Afterwards, they proposed a multiobjective offloading strategy based on NSGA-III (Nondominated Sorting Genetic Algorithm III) and realized the optimization of time consumption and resource utilization in MEC. Li et al. [35] discussed how to divide and allocate the divisible applications and proposed a computation offloading method according to the capabilities of available resources, thereby enabling the divisible applications to be executed in less time. Chen and Hao [40] investigated the computing offloading problem for tasks generated by some innovative applications. They divided the optimization problem into task placement subproblem and resource allocation subproblem. Then, they proposed an efficient offloading method, which can optimize the energy consumption and delay of tasks jointly.

These studies mainly focused on the general applications, but WAs are quite different and more complex than general applications; thus, we should put more emphasis on how to make a suitable offloading strategy for WAs in MEC. Although the research on WAs is more challenging, it is also very important, which is promising to solve the computing offloading problem of complex tasks consisting of time-constrained workflows and concurrent workflows. And there have been some related researches and achievements in recent years.

Chen et al. [23] studied the cost problem of scientific workflows. They hold the view that the high latency caused by transmission and storage is the reason for the high cost. To cope with the problem, they proposed a data placement method based on the genetic algorithm and the particle swarm optimization algorithm, which can reduce the cost of scientific workflows greatly. Zhu et al. [24] focused on how to build an efficient offloading mechanism and proposed a deep Q-learning based offloading algorithm, which greatly shortened the energy consumption and the execution time of service workflow. Huang et al. [39] designed an offloading method based on nondominated sorting differential evolution. The objective of them is to optimize the energy consumption of MDs while meeting the deadline constraints. Wang et al. [41] proposed a particle swarm optimization based offloading algorithm to optimize the energy consumption of MDs. And in order to solve the problem of delay in the process of task transmission, they built a decision model of mobile cloud workflow and optimized the energy consumption of MDs while meeting the deadline constraints.

In our previous work, we have done some studies on computing offloading in MEC environment [34] and have proposed a multiobjective optimization of energy consumption and cost for Was [48]. Different from the existing research, we use a new WAs' model to evaluate complex tasks, which consist of the time-constrained workflows and the concurrent workflows. And a multiobjective joint optimization of energy consumption and time consumption of MDs as well as the resource utilization of cloudlets for multiconstraint WAs in MEC is proposed.

3. Multicloudlet Mode and Problem Formulation

In this section, the system mode of multiconstraint WAs in multicloudlet environment is proposed to evaluate the energy consumption and time consumption of MDs as well as the resource utilization of cloudlets. Then, our problems and goals are formulated. The description of some key parameters in system mode is shown in Table 1.

3.1. The Multicloudlet System Mode. The network architecture of MEC is shown in Figure 1 [49]. Cloud is usually seen as a data center, which has nearly inexhaustible computing resources but far from MDs. Cloudlet is an independent structure in network edge, which has more computing resources than MDs and can communicate with data center through WAN. In multicloudlet environment, MDs can select a suitable server and communicate with this server through LAN. In addition, MDs can also interact with cloud via WAN directly.

Because of the explosion of the Internet, MDs often need to deal with a large amount of complex tasks and most of them have concurrent constraints or time constraints. Therefore, how to build a suitable model for complex tasks is also one of the critical problems. In this article, the complex tasks generated by MDs is modeled as multiconstraint WAs and represented by a directed acyclic graph (DAG), denoted as $G_f = (V_f, D_f, S_f)$, where $V_f = \{v_{1,f}, v_{2,f}, \ldots, v_{n,f}\}$ represents the amount of data that each task in the f-th WAs should be calculated and $D_f = \{r(r_{i,f}, r_{j,f}), d_{i,j}\}$ represents the relationship between subtasks in f-th WAs, which clearly specifies the execution order between two subtasks. For example, there is a WA set as $\{v_1, v_2, v_3, v_4\}$, and the relationship between its subtasks is expressed as $r\{(v_1, v_2), (v_1, v_3), (v_2, v_4)\}$, namely, task v_1 is the direct and the only precursor of task v_2 and task v_3, denoted as $\mathrm{Pre}_2 = \{v_1\}$ and $\mathrm{Pre}_3 = \{v_1\}$. Thus, task v_2 and task v_3 can only be computed after task 1 has been executed. Similarly, task v_4 must wait for task v_2, its predecessor, to be completed. In addition, the number of the tasks $n \in \{1, 2, \ldots, N\}$ and the WAs' number $f \in \{1, 2, \ldots, F\}$.

$S_f = \{s_{1,f}, s_{2,f}, \ldots, s_{n,f}\}$ represents the offloading strategy of the f-th WA. Generally, in multicloudlet environment,

Table 1: Key terms and descriptions.

Parameter	Meaning
L	Queue length
F	The number of WAs
B	Bandwidth of communication link
E	Computing ability of the terminal
N	The number of subtasks in each WA
M	The number of the virtual machines in cloudlet
I	The label of the cloudlet
U_I	The number of tasks executed by the I-th cloudlet
St	Situation set of offloading strategy
Suc	The collection of successor tasks
Pre	The collection of precursor tasks
B_{Se}	Bandwidth of transmission under same system
B_{De}	Bandwidth of transmission under different system

most part of tasks will be offloaded. Hence, in order to facilitate the evaluation of resource utilization, we do not consider the situation that tasks are directly executed on MDs without offloading. In view of this, we define that the label of the cloudlet is tagged from 2 to C and the offloading strategy $s_{n,f} \in \{2, 3, \ldots, C, C+1\}$. When $s_{n,f} = 2, \ldots, C$, the task will be offloaded to cloudlet for execution by LAN. Then, the task can obtain more resources in a relatively short time, which is also the offload strategy for the most part of tasks. Similarly, when $s_{n,f} = C + 1$, the task will be offloaded to cloud via WAN for computation and the task can obtain nearly infinite computing resources, but it also consumes a lot of time for transmission.

3.2. Time Consumption Mode. In MEC environment, time consumption mainly consists of three parts, namely, transmission time T_{tra}, queueing waiting time T_{que}, and execution time T_{exe}. Next, we will analyze these three aspects in details.

3.2.1. Transmission Time. For most WAs, a certain amount of data will be generated after the task is completed, and the data needs to be transmitted to its successor tasks, what will cause transmission delay. For each task, the transmission delay depends on its own offloading strategy and the offloading strategy of its successor tasks. For example, the successor tasks of v_i are defined as Suc_i, and $\mathrm{Suc}_i = \{v_j, v_k\}$. Hence, v_i has two successor tasks v_j and v_k. After v_i is completed, some data will be transferred to both v_j and v_k. The transmission time from task v_i to tasks v_j and v_k can be calculated as

$$T_{\mathrm{tra}}\left(v_i, \left(v_j, v_k\right)\right) = \left(\frac{d_{i,j}}{B_{i,j}} + \frac{d_{i,k}}{B_{i,k}}\right), \tag{1}$$

where $d_{i,j}$ is the size of data transmitted between tasks v_i and v_j, and $B_{i,j}$ is the bandwidth of the network used to transmit tasks, but it will change as the offloading path of two interrelated tasks changes. For tasks v_i and v_j, when the strategy s_i is the same as s_j, we express this situation as St = 1, and because the transmission process is carried out on the same server, the transmission speed will be very fast. We assume the bandwidth $B_{i,j} = \infty$ at this time. When both s_i and $s_j \in (2, 3, \ldots, C)$, but $s_i \neq s_j$. We define this situation as St = 2, and $B_{i,j} = B_{\mathrm{Se}}$. Similarly, when one of s_i or $s_j \in (2, 3, \ldots, C)$, and the other equals $C + 1$, the situation is denoted as St = 3. At this point, the data will be transmitted through WAN, but the transmission speed will be lower than that in the second case due to the long distance, and the bandwidth $B_{i,j} = B_{\mathrm{De}}$. The selection of $B_{i,j}$ can be described as

$$B_{i,j} = \begin{cases} \infty, & \mathrm{St} = 1, \\ B_{\mathrm{Se}}, & \mathrm{St} = 2, \\ B_{\mathrm{De},} & \mathrm{St} = 3. \end{cases} \tag{2}$$

3.2.2. Queueing Waiting Time. Cloudlet can provide less time consumption and less energy consumption network environment for tasks than MDs, but its capacity is limited, it is also one of the most critical issues of MDs. When the number of tasks which have been offloaded to the cloudlet is larger than the cloudlet's capacity, late tasks can only be computed after all of its precursor tasks have been computed, which also leads to queueing delay. The capacity of the cloudlet is evaluated by the number of virtual machines (VMs) on it, which is expressed as M. In this paper, we assume that the number of VMs per cloudlet is close, but their latency and computing capacity is different. Based on the queueing theory, we assume that the arrival time interval of tasks as δ, and the time served by cloudlet as η [50]. The values of δ and η are subjected to a negative exponential distribution. So, the probability that the cloudlet is inactive can be expressed as

$$P_{\mathrm{off}} = \left[\sum_{L=0}^{M-1} \frac{\lambda^L}{L!} + \frac{\lambda^M}{M!\left(1 - \lambda_M\right)}\right]^{-1}, \tag{3}$$

where $\lambda = \delta/\eta$ denotes the usage of cloudlet, and $\lambda_M = \lambda/M$. Therefore, when a cloudlet works steadily, the probability that the queue length equals L can be expressed as

$$P_L = \begin{cases} \dfrac{P^L}{L!} \cdot P_{\mathrm{off}}, & L \leq M, \\ \dfrac{\lambda^L}{M! M^{L-M}} \cdot P_{\mathrm{off}}, & L > M. \end{cases} \tag{4}$$

When $L > M$, the waiting time of tasks in cloudlet will be considered, and the average queue length can be expressed as

$$L_q = \sum_{L=M+1}^{\infty} (L - M) P_L + \lambda = \frac{P_{\mathrm{off}} \cdot \lambda^M}{M!} \cdot \sum_{L=M}^{\infty} (L - M) \lambda_M^{L-M} + \lambda. \tag{5}$$

In summary, the waiting time of WAs can be calculated by

$$T_{\mathrm{que}} = \frac{L_q}{\delta} - \frac{1}{\eta}. \tag{6}$$

3.2.3. Execution Time. The execution time of a task on the target system is composed of the computation time and the network delay. When tasks are offloaded to cloudlet for

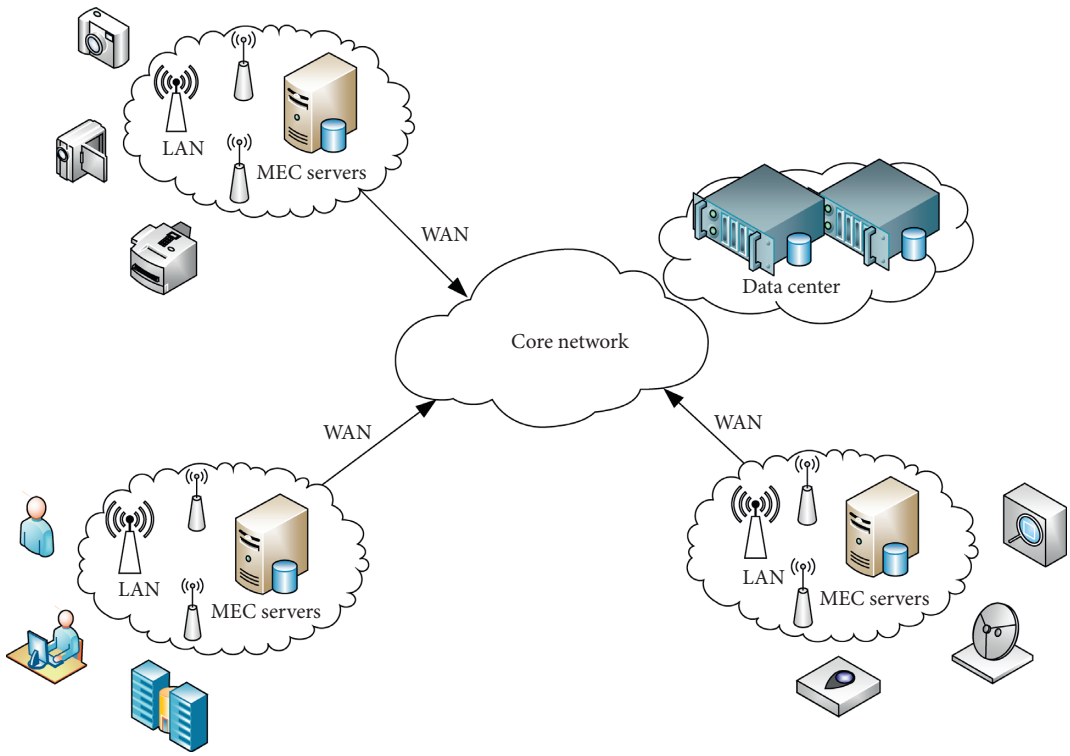

Figure 1: Multicloudlet architecture.

execution, corresponding network delay will be generated, which is expressed as D_{LAN}. Similarly, when task are offloaded to cloud for execution, the delay is expressed as D_{WAN}. Thus, the execution time of the task can be described as

$$T_{\mathrm{exe}}\left(v_{n,f}\right) = \begin{cases} W_q + \dfrac{v_{n,f}}{E_{\mathrm{cl}}} + D_{\mathrm{LAN}}, & s_{n,f} = 2 \text{ or } 3, \text{ or } \ldots \text{ or } C, \\ \dfrac{v_{n,f}}{E_c} + D_{\mathrm{WAN}}, & s_{n,f} = C + 1, \end{cases} \tag{7}$$

where E_{cl} and E_c are the processing power of the cloudlet and cloud, respectively. Hence, full-time consumption on task execution can be described as

$$T_{\mathrm{total}}\left(v_{n,f}\right) = \begin{cases} \left(T_{\mathrm{tra}} + T_{\mathrm{que}} + T_{\mathrm{exe}}\right)_{\left(v_{n,f}\right)}, & s_{n,f} = 2 \text{ or } 3, \text{ or } \ldots \text{ or } C, \\ \left(T_{\mathrm{tra}} + T_{\mathrm{exe}}\right)_{\left(v_{n,f}\right)}, & s_{n,f} = C + 1. \end{cases} \tag{8}$$

3.3. Energy Consumption Mode. Through the analysis of the time consumption of WAs, it is easier to get its total energy consumption. Refining to each task, its energy consumption is mainly composed of execution energy and transmission energy. If tasks are offloaded to cloudlet, queueing waiting energy will also be considered. Therefore, full energy consumption of MDs can be described as

$$E_{\text{total}}\left(v_{n,f}\right) = \begin{cases} \left(T_{\text{tra}} + T_{\text{que}} + T_{\text{exe}}\right)_{\left(v_{n,f}\right)} \times P_I, & s_{n,f} = 2 \text{ or } 3, \text{ or } \ldots \text{ or } C, \\ \left(T_{\text{tra}} + T_{\text{exe}}\right)_{\left(v_{n,f}\right)} \times P_I, & s_{n,f} = C + 1, \end{cases} \tag{9}$$

where P_I is the computing power when the MD is idle; at this time, tasks are offloaded to the end-system outside the MD for execution, and MD only needs a small power to wait for the task to be completed.

3.4. Resource Utilization Mode. We assume that the resources of the cloudlet are limited. Therefore, how to improve the resource utilization of the cloudlet is still a problem. As shown in (10), the resource utilization of the $I-th$ cloudlet R_I is related to the number of tasks U_I executed by this cloudlet, namely, the more effective the tasks on the cloudlet, the higher the resource utilization rate. But when the number of tasks is equal to the capacity of the cloudlet, which is usually expressed by the number of VMs on it (i.e., M), the resource utilization will reach 1 and it will stay the same.

As shown in (11), the label of the cloudlet I is tagged as $\{2, 3, \ldots, C\}$. Similar to the description of offloading strategy $s_{n,f}$ in Section 3.1, the label of the cloudlet is tagged from 2 to C, namely, there are $C-1$ cloudlets in all. And as equation (12) shows, U_I represents the number of tasks executed by the $I-th$ cloudlet. We define the quantity of WAs as f and each WAs contains n subtasks; thus, the max value of U_I equals $f \times n$ while the minimum value equals 0.

$$R_I = \begin{cases} \dfrac{U_I}{M}, & U_I < M, \\ 1, & U_I \geq M, \end{cases} \tag{10}$$

$$I \in \{2, 3, \ldots, C\}, \tag{11}$$

$$U_I \in \{0, 1, \ldots, f \times n\}. \tag{12}$$

3.5. Problem Formulation. The purpose of computation offloading in MEC environment is to optimize the time consumption and energy consumption of MDs as well as the resource utilization cloudlets under the deadline constraints set by mobile users. Our optimization problems can be described as

$$\begin{aligned} & s_n \in \{2, 3, \ldots, C+1\}, \\ \text{Min } & T_{\text{total}}\left(v_{n,f}\right), \quad \forall f \in \{1, 2, \ldots, F\}, \\ \text{Min } & E_{\text{total}}\left(v_{n,f}\right), \quad \forall f \in \{1, 2, \ldots, F\}, \\ & \text{Balance } R_I, \quad \forall I \in \{2, 3, \ldots, C\}, \\ \text{s.t. } & T_{\text{total}}\left(v_{n,f}\right) \leq T_{\text{ddl}}, \quad \forall f \in \{1, 2, \ldots, F\}. \end{aligned} \tag{13}$$

4. Multiobjective Optimization Algorithm for Complex Tasks in Multicloudlet Environment

In this section, we will describe the details of MOCME. Firstly, we will give a brief description of the concept of Nondominated Sorting Genetic Algorithm II (NSGA-II), and the three parts, gene coding, computation offloading strategy, and the structure of multiconstraint, WAs are redefined. Then, the generation method and the mechanism of multiconstraint WAs are described in details. Finally, the basic steps of MOCME and corresponding pseudocode are explained.

4.1. Redefinition. In this paper, our goal is to optimize the time consumption and energy consumption of MDs as well as resource utilization of the cloudlets. Therefore, we choose NSGA-II as our basic algorithm. Compared with other optimization algorithms, NSGA-II has better convergence and faster running speed in solving multiobjective optimization problem, and its complexity is optimized on the basis of NSGA. However, it is still a problem to optimize multiconstraint WAs in multicloudlet environment. Thus, we propose a new method to solve the problem, named MOCME.

The first step of MOCME is to redefine some basic variables and methods, which is also one of the critical steps of MOCME. After gene coding, the task offloading model can be easily constructed and the optimization of resource utilization will be more effective through the simplification of offloading strategy. Meanwhile, by decomposing the multiconstraint WAs into two parts (i.e., unordered WAs and ordered WAs), the complexity of the WAs is also simplified. The details of redefinition can be described as follows :

Redefinition 1. Gene coding: we use the idea of genetic algorithm to reconstruct the complex tasks consisting of time-constrained workflows and concurrent workflows. Firstly, every subtask in a WA will be numbered with integer quotes and starts from $1, 2, \ldots, n$. Then, every WAs will be represented by chromosomes, and the corresponding subtasks will be represented by genes, namely, several genes constitute a chromosome, which is what we call WA and each population is consisted by several chromosomes. As shown in Figure 2, a workflow can be transformed into a chromosome according to the tasks' execution sequence, and the number of each gene represents their offloading strategy.

Redefinition 2. Computational offloading strategy: in multicloudlet environment, most tasks will be offloaded to cloudlet or cloud for execution. Furthermore, the resource utilization of the cloudlet is also one of our goals, so we redefine the offloading path

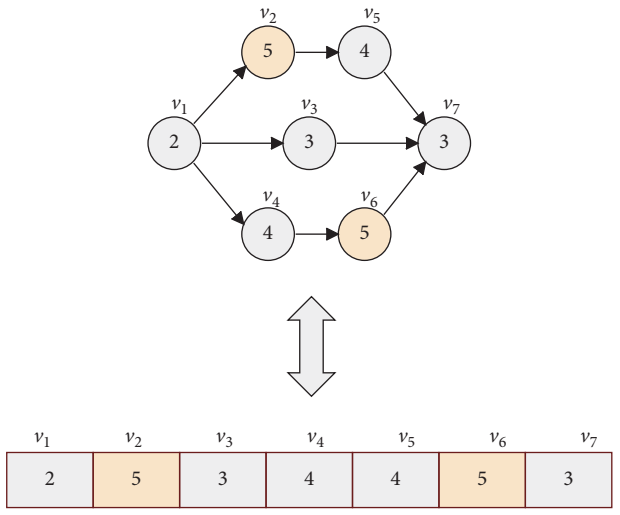

Figure 2: Gene coding process.

$S_{new} \in \{2, 3, \ldots, C, C+1\}$. It does not consider the situation that tasks are directly executed on MDs, and the result can better show the optimization of resource utilization.

Redefinition 3. Connector $\otimes$ for unordered workflow applications (UWs): for an UW, we use symbol $\otimes$ to link every subtask within it and these tasks can be executed concurrently. Then, an UW can be represented as $V_{UW} = \{v_1 \otimes v_2 \otimes \cdots \otimes v_n\}$. Each UW can be constructed by a random combination of their subtasks, which has quite a variety of possibilities, and it is similar to the creation process of multiconstraint WAs, the specific construction details are described in Section 4.2.

Redefinition 4. Connector $\oplus$ for ordered workflow applications (OWs): compared with UWs, the construction of OWs will be a little complex and the tasks in it cannot be executed concurrently. For example, there are four related tasks $\{v_1, v_2, v_3, v_4\}$, whose execution must be sequential, whether there is an idle cloudlet or not. Thus, an OW with four subtasks can be expressed as $\{v_1 \oplus v_2 \oplus v_3 \oplus v_4\}$.

We use the recursive method to build from the end of the WA. Firstly, the subtasks without precursors will be selected as the end nodes. Then, we can get the predecessor tasks of the end nodes by traversing all of the tasks. Finally, we just need to filter out the nodes that have been built and repeat the above traversal operation. By constructing OWs, we can divide the types of each WA in multiconstraint WAs, and its model can be easily established.

4.2. The Mechanisms of Multiconstraint Workflow Applications. In order to evaluate various types of complex tasks generated in MDs, we propose a new workflow mode, named multiconstraint WAs. Based on the relationship between genes and chromosomes in GA, each WA is temporarily designated by genes during the construction of WAs, and the chromosome at this time is equivalent to the whole population. As shown in Figure 3, we assume that

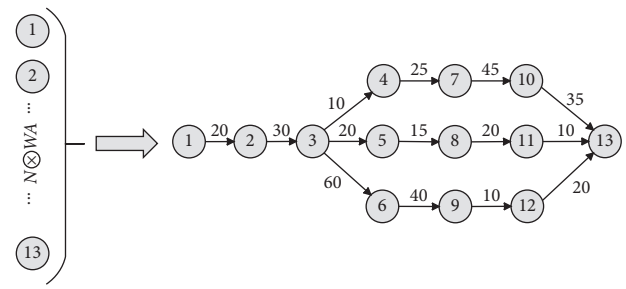

Figure 3: Composition of multiconstraint workflow applications.

multiconstraint WAs are consisted of $n \oplus$ task and $n \otimes$ task, each WA as a gene, and these 13 WAs constitute a population, expressed as $V_{all} = \{V_1 \otimes V_2 \cdots \otimes V_{13}\}$.

In Figure 3, there is no strong connection between each WA, which is similar to the subtasks in the UWs. There are also many other architectures of this kind of WAs, we only show one of them.

Based on the description of UWs and OWs, a complete multiconstraint WAs can be constructed by Algorithm 1. First of all, we need to use the recursive method to integrate the WAs for the first time (Line 2–6). Then, each WA needs to be partitioned, that is, UWs and OWs are separated (Line 7–13). After partitioning is completed, the tasks in each OWs are still discrete, so we need to integrate these subtasks for a second time to make them become a complete WA (Line 15–19), namely, each WA has been partitioned. Next, we need to integrate the divided WA for the third time. Generally, the relationship between two WAs is also unordered, and the basic idea of constructing a multiconstraint WAs is similar to the integration of UWs. Thus, the WAs will be investigated on a first come, first service basis (Line 20–22).

4.3. The Basic Steps of MOCME. Similar to NSGA-II, MOCME can be divided into five steps: Initialization, Selection, Crossover, Mutation, and the Calculation of Crowding Distance. Next, we will describe these five steps in detail and give a review of the whole method in the end.

4.3.1. Initialization. Before the algorithm is executed, we need to do some initiate operations. For example, the subtasks input need to be converted into executable WAs in order to construct an analyzable parent population for subsequent genetic operations, namely, to divide and integrate the multiconstraint WAs. Moreover, some key parameters such as iteration number G_{max} and the deadline T_{ddl} also need to be set firstly.

4.3.2. Selection. Selection is a critical step in MOCME, parent population and the progeny population will be merged and a new hybrid-population Q will be generated, denoted as $Q = P_{par} + P_{std}$. According to the crowding distance of each individual in Q, the better individuals will be selected to enter the next generation. The details of crowding distance are described in Section 4.3.5.

Input: The collection of tasks V, tasks' relationship D;
Output: Available multiconstraint workflow applications V_f;
(1) **for** Every workflow application V_f **do**
(2) **for** Every task per workflow application v_n unready **do**
(3) **if** $\mathrm{Suc}_n = 0$ or $\forall \mathrm{Suc}_n$ = ready **then**
(4) $v_n \longrightarrow$ ready
(5) **end if**
(6) **end for**
(7) **for** $v_n \in V_f$ **do**
(8) **if** $\exists \mathrm{Suc}_n ! = 0$ **then**
(9) $V_f \longrightarrow$ ordered workflow application V_{ow}
(10) **else**
(11) $V_f \longrightarrow$ unordered workflow application V_{uw}
(12) **end if**
(13) **end for**
(14) **end for**
(15) **for** $\forall V_{\mathrm{uw}}$ **do**
(16) **for** $\{v_2, \ldots, v_N \in V_{\mathrm{uw}}\}$ **do**
(17) $\mathrm{Pre}_{n+1} = v_n$
(18) **end for**
(19) **end for**
(20) **for** $\{V_2, \ldots, V_F\}$ **do**
(21) $\mathrm{Pre}_{f+1} = V_f$
(22) **end for**
(23) **return** V_f

Algorithm 1: Construction of multiconstraint workflow applications.

4.3.3. Crossover. Crossover is one of the classical optimization methods, which can prevent the algorithm from falling into local convergence. As shown in Figure 4, the same part of the chromosome will be retained while the different part will be crossed. We can easily analyze that the same gene fragment can be regarded as a better part. By crossover operation, the better genes can be retained, and the algorithm will not fall into local convergence.

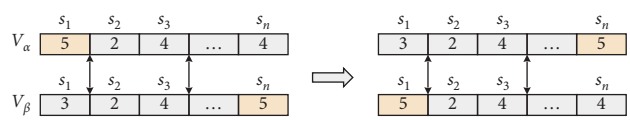

Figure 4: Crossover.

4.3.4. Mutation. Mutation is another commonly used optimization method. As shown in Figure 5, s_3 changes its offloading strategy from 5 to 3 through mutation, which means the cloudlet will provide services for s_3 that should have been offloaded to cloud. Every particle has the same mutation probability and their offloading strategy is all likely to be changed in each iteration.

By controlling the mutation probability ω, the algorithm can be prevented from falling into local convergence while the diversity of population is maintained. More significantly, when we face a huge amount of tasks, the convergence speed of the algorithm can be improved by directed mutation.

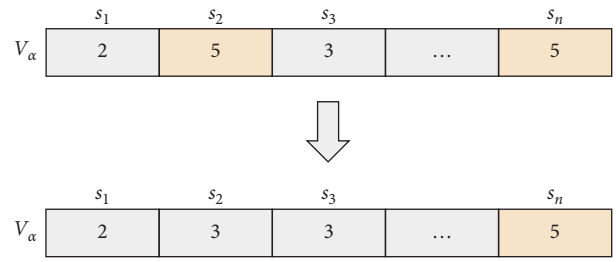

Figure 5: Mutation.

4.3.5. Crowding Distance. In NSGA-II, crowding distance is proposed to replace the shared parameter in NSGA, which greatly reduces the complexity of the algorithm. In MOCME, crowding distance is also the basis for updating particles. The calculation method of crowding-distance is shown in (14) and (15):

$$C_x = \varepsilon \times E_x + \tau \times T_x + \iota \times R_x, \tag{14}$$

$$X_x = \frac{X_{\max} - X_{\mathrm{now}}}{X_{\max} - X_{\min}}. \tag{15}$$

In equation (14), the objective is the hybrid-population Q consists of initial population V_{par} and progeny population V_{std}. E_x, T_x, and R_x represent energy consumption, time consumption, and resource utilization, they are also the goals we need to evaluate. The coefficient is the weight of each objective in the evaluation, which can be changed as our needed and usually $\varepsilon = \tau = \iota = 1$. Equation (15) is the

concrete method for evaluating the objective, where $X \in \{E, T, R\}$.

4.3.6. Algorithm Description. Algorithm 2 shows the pseudocode of MOCME. First of all, the input target population needs to be reconstructed by Algorithm 1 to form a multiconstraint WAs, which has a more regular and simple structure, and the relationship between the subtasks is more specific (Line 1). After initial operation such as modeling and coding, the better individuals are selected according to their crowding distance L_{st} to generate a offspring V_{std}. The crowding distance L_{st} can be calculated according to equation, (14) and (15) (Line 4–6). And each particle in offspring V_{std} is selected one by one from the parent population V_{par}. When the number of particles in offspring population reaches the maximum number N, the selection stops (Line 7–13). After the selection of V_{par}, the offspring V_{std} will be further updated and become a qualified offspring by crossover and mutation (Line 14–15). Then, the hybrid-population Q is formed by adding the first generation to its offspring (Line 16). Next, we need to do a second selection operation, which is one of the most important steps in MOCME. Through the selection of hybrid-population Q, we can update the parent population V_{par}. Under such optimization repeatedly, the final paternal population V_{par} is the ideal population we need (Line 17–26).

5. Experimental Evaluation

In this section, the evaluation of the experiments is described in details. Firstly, the experiment environment and some key parameters are set up. Then, in order to evaluate MOCME effectively, several groups of comparative experiments are proposed. Finally, the test results will be shown, and we will analyze the result by comparing different methods.

5.1. Experimental Settings. Our experiment is conducted in MEC environment with three cloudlets. We consider that most of the tasks will be offloaded; in this way, MDs will remain idle and they have a low standby power. In addition, we have different settings for the three cloudlets and their computing capacity and network latency are different from each other, which is similar to the actual situation, and it also increases the comparability of our experiments. Furthermore, we assume that cloud contains abundant computing resources but bears a high time consumption for transmission. Some critical parameters are shown in Table 2.

The changes of the network environment will cause a huge impact on the results of the method. Thus, we choose the waiting time of the task on the cloudlet as a variable to evaluate the performance of MOCME and other methods in a different network environment. In our experiment, each workflow contains 13 subtasks, and the number of WA is set to 4–6. In addition, each method will be tested more than 50 times under the same conditions. The method is implemented by MATLAB, on a physical machine with 2 Intel Core i5-6500U 3.20 GHz processors and 8 GB RAM and the operating system is Win7 64.

5.2. Comparative Method. In order to evaluate the method we proposed objectively, we construct several other methods different from MOCME. The details of these methods are described as follows:

All Random Offloading (ARO). All subtasks in WAs will be offloaded randomly, the task can be offloaded to any part of $\{2, 3, \ldots, C + 1\}$, regardless of energy consumption and other goals, named as ARO. This method is relatively simple but may have a large accidental error.

First Come First Service (FCFS). Each subtask in WA will be offloaded sequentially, following the first come first service principle, named as FCFS. For example, there is a WA denoted as $V = \{v_1, v_2, v_3, v_4\}$ whose offloading strategy can be expressed as $S = \{2, 3, 4, 5\}$.

All Offloaded to Cloudlet (AOCL). Every task will be offloaded to the cloudlet for execution, and the tasks can select one of the cloudlets randomly, named AOCL. At this time, each task can schedule more resources in cloudlet, but it may also result a long queue waiting beyond cloudlet.

MOCME. The offloading strategy of WAs is selected according to the three objectives, i.e., energy consumption and time consumption of MDs as well as the resource utilization of cloudlet while meeting the deadline constraints, named MOCME.

In these three comparison methods (i.e., ARO, FCFS, and AOCL), a large part of the tasks will be offloaded to the cloudlet for execution, because normally the comprehensive performance of cloudlet is better than that of MDs or cloud which further increases the difficulty but ensures that the optimization of our method is effective compared with the general offloading method.

Moreover, the working mechanism and expected results of these methods are very different; thus, they may have different optimization capabilities in terms of energy consumption, time consumption, and resource utilization. Furthermore, with the impact of network environment, such as network congestion, the performance of the three comparison methods will be significantly affected. Therefore, the optimization effect of MOCME in different network environment can be better evaluated.

5.3. Performance Evaluation. In this section, energy consumption, time consumption, and resource utilization will be evaluated, and we will construct a comparative experiment by increasing the number of WAs and queueing waiting time of the cloudlet. Furthermore, the comparative experimental steps of these four methods: ARO, FCFS, AOCL, and MOCME will be described in details.

5.3.1. Time Consumption Evaluation. The three parts: task transmission time, queueing waiting time, and execution time constitute the whole time consumption of MDs, which are calculated by equations (1), (6), and (7), and the total time consumption is derived from equation (8). The

Input: Target population $G_f = (V, D, S)$, Iteration times $G_{\max}$, Deadline T_{ddl};
Output: Optimal location $S = (s_1, s_2, \ldots, s_n)$, Time consumption of MDs, Energy consumption of MDs, Resource utilization of the cloudlets;

(1) Use **Algorithm 1** to initialize the parent population V_{par}
(2) $g = 1, \text{num} = 1$
(3) **while** $(g \leq G_{\max})$ **do**
(4) **for** Every particle $v_n \in V_{\text{par}}$ **do**
(5) Bring v_n into (14) and (15) to get its crowding distance L_{st}
(6) **end for**
(7) **for** Every particle $v_n \in V_{\text{par}}$ **do**
(8) **while** $(\text{num} \leq N)$ **do**
(9) Select $V_{\text{par}} \longrightarrow V_{\text{std}}$
(10) num = num + 1
(11) **end while**
(12) num = 1
(13) **end for**
(14) $V_{\text{std}} = \text{Mutation } V_{\text{std}}$
(15) $(V_{\text{par}}, V_{\text{std}}) = \text{Crossover } (V_{\text{par}}, V_{\text{std}})$
(16) $Q = V_{\text{par}} + V_{\text{std}}$
(17) **for** Every particle $v_n \in Q$ **do**
(18) Bring Q into (14) and (15) to get its crowding distance L_{rd}
(19) **end for**
(20) **for** Every particle $v_n \in Q$ **do**
(21) **while** $(\text{num} \leq N)$ **do**
(22) Select $Q \longrightarrow V_{\text{par}}$
(23) num = num + 1
(24) **end while**
(25) num = 1
(26) **end for**
(27) $g = g + 1$
(28) **end while**
(29) **return** S, Time, Energy, Resource utilization

Algorithm 2: MOCME.

Table 2: Parameter settings.

Parameter	Value
Latency of WAN	30 ms
Latency of LAN 1	0.8 ms
Latency of LAN 2	1.2 ms
Latency of LAN 3	1 ms
Power when MDs in idle	0.001 W
Transmission power of MDs	0.1 W
Computing capacity of the first cloudlet	1900 MHz
Computing capacity of the second cloudlet	2000 MHz
Computing capacity of the third cloudlet	2200 MHz
Computing capacity of the cloud	3000 MHz

comparison of different methods in terms of time consumption is shown in Figure 6.

It can be seen that with the increasing of the queueing waiting time, the time consumption of all methods will be increased significantly. However, MOCME has a better optimization result performance for time consumption, especially when it takes a long waiting time.

For example, when the number of WAs equals 6, both AOCL and MOCME consume less time than ARO and FCFS when the task only requires 10 ms for waiting. This is because the tasks can be transferred to the cloudlet for execution in a short time compared with the high time consumption of tasks transferred to cloud. But when the waiting time increases to 50 ms or 100 ms, the time consumption will be increased dramatically in cloudlet, and AOCL will consume more time than ARO and FCFS while the time consumption of MOCME will still be in a low level by adjusting its offloading strategy. Furthermore, AOCL does not consider other goals comprehensively and has a huge uncertainty. Therefore, compared with the other three methods, MOCME is a satisfactory strategy.

In terms of the number of WAs, MOCME still has a great performance. However, under the same queueing waiting time, the optimization capability of MOCME is not changed a lot with the increasing of the WAs number. Thus, we can get that the queueing waiting time of the tasks on cloudlet may play a more important role in the optimization performance compared to the number of WAs.

5.3.2. Energy Consumption Evaluation. Similar to the time consumption, the tasks' energy consumption mainly consists of three aspects: transmission consumption, queue standby consumption, and execution consumption, which can be calculated by (9). The energy consumption comparison of the four methods is shown in Figure 7. It can be seen that MOCME still has a good optimization

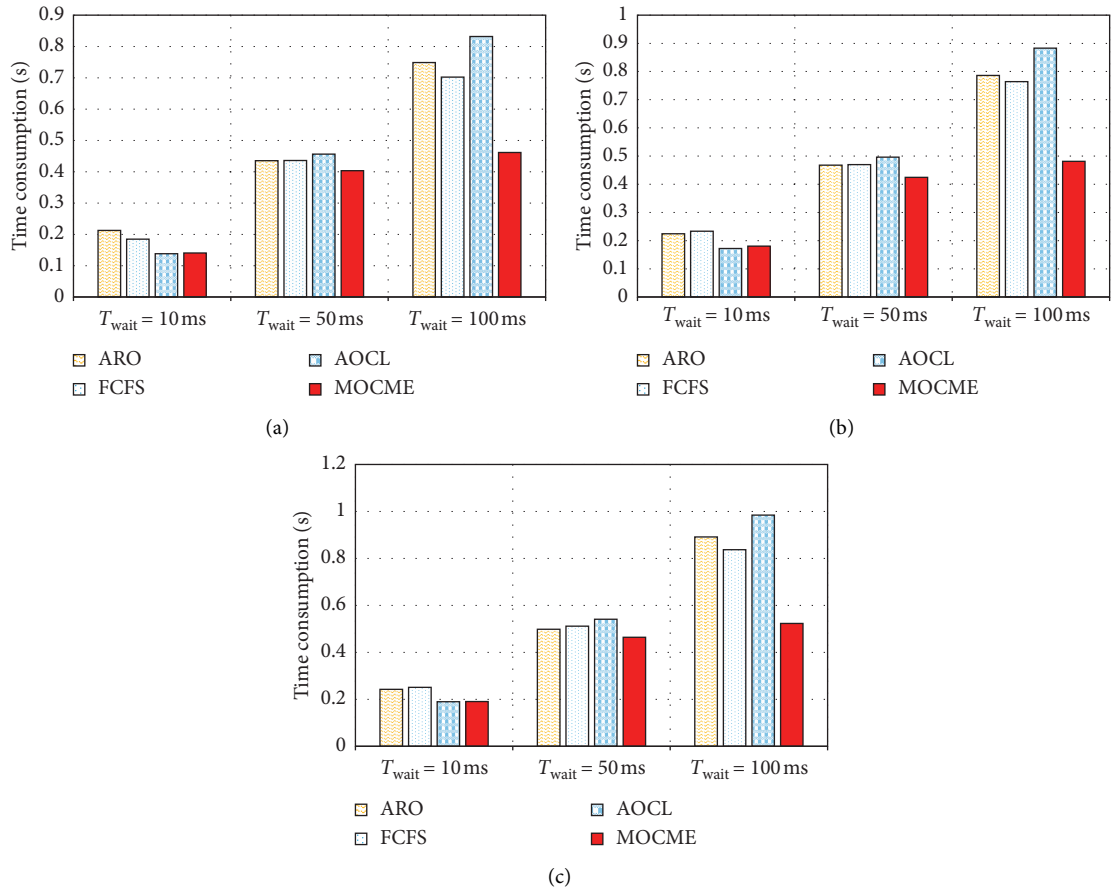

Figure 6: Average time consumption of MDs. (a) Number of WAs = 4. (b) Number of WAs = 5. (c) Number of WAs = 6.

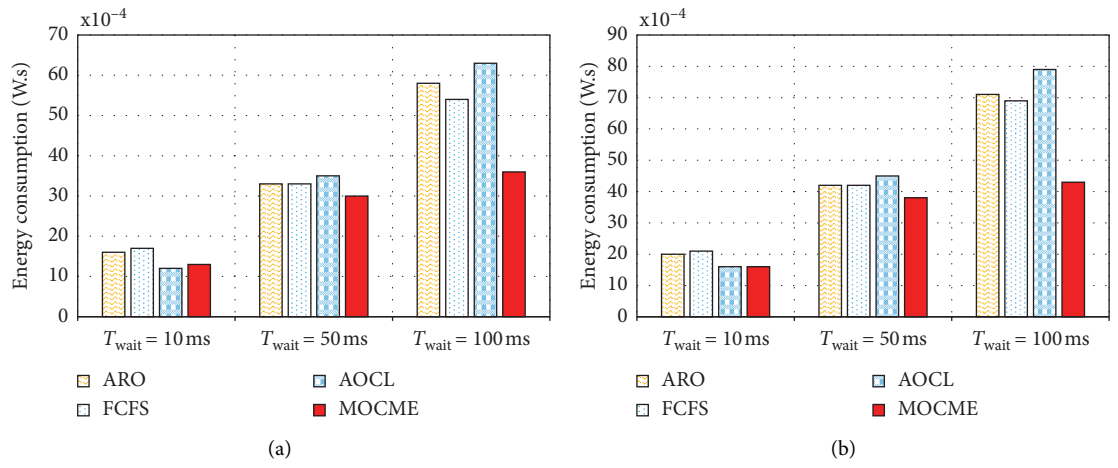

Figure 7: Continued.

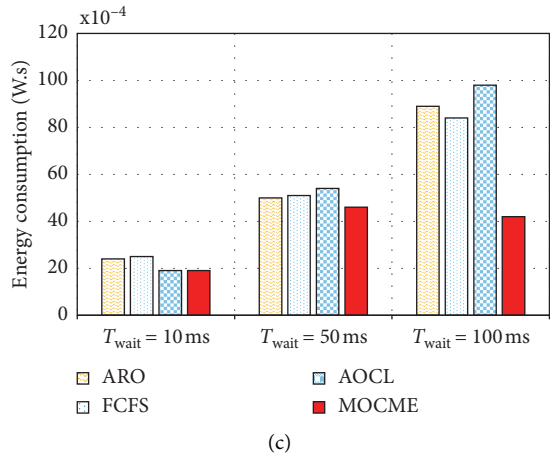

Figure 7: Average energy consumption of MDs. (a) Number of WAs = 4. (b) Number of WAs = 5. (c) Number of WAs = 6.

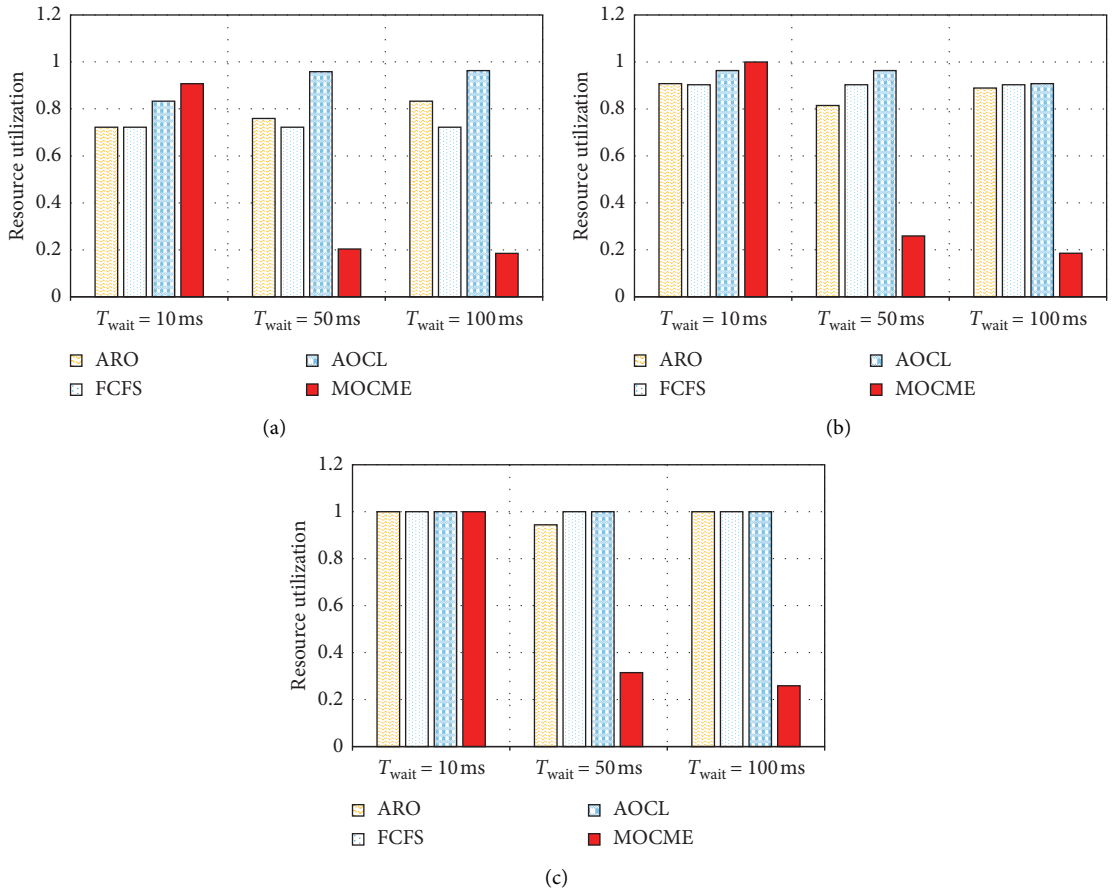

Figure 8: Average resource utilization of the cloudlets. (a) Number of WAs = 4. (b) Number of WAs = 5. (c) Number of WAs = 6.

performance in high delay network environment. Moreover, as the number of WA increases, the optimization performance of MOCME will also improve effectively. Therefore, compared with the performance in time consumption, the optimization performance in energy consumption of MOCME is not only related to the queueing waiting time but also related to the number of WA.

5.3.3. Resource Utilization Evaluation. Usually, we assume that the computing resources of the cloudlet are limited and the waste of the resources will lead to a further increase in costs. Therefore, how to improve the resource utilization of the cloudlet is one of the key factors we are concerned about.

As shown in Figure 8, we can see the resource utilization of ARO, FCFS, AOCL, and MOCME in different situations,

and the resource utilization of the cloudlet can be obtained by (10). We can knew that the resource utilization of MOCME can reach a relatively high level when the queueing waiting is time at a low standard, 10 ms. However, when the queueing waiting time reaches 50 ms, the resource utilization of MOCME will drop sharply. This is because queueing at this time will take a lot of time, even more than the time consumption of offloading tasks to the cloud, the majority of the tasks will be offloaded to the cloud for execution. When the queueing waiting time reaches 100 ms, the resource utilization of the cloudlet will be further decreased.

6. Conclusion

In this paper, we have studied the energy consumption, time consumption, and resource utilization issues for complex tasks in MEC. To solve the problem, a multiconstraint workflow mode is designed and a multiobjective optimization algorithm is proposed, named MOCME. Specifically, we have redefined some key quantities and described the construction process of multiconstraint WAs and the operation process of MOCME in detail. Finally, a series of experiments and detailed data analysis are proposed to prove our method is efficient and efficiency.

In future work, we will focus on the impact of MDs' mobility on computation offloading in MEC [9]. For one thing, the network changes brought by MDs' mobility will be analyzed. For another thing, the changes in energy consumption, time consumption, and resource utilization caused by network changes will be studied.

Data Availability

The data used to support the findings of this study are included within the article. Readers who are interested in our research can contact our coauthor Bohai Zhao (amebhzhao@foxmail.com).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Science Foundation of China (Grant no. 61902133), the Natural Science Foundation of Fujian Province (Grant no. 2018J05106), the Education and Scientific Research Projects of Young and Middle-aged Teachers in Fujian Province (JZ160084), and the Scientific Research Foundation of Huaqiao University under Grant no. 14BS316, Quanzhou Science and Technology Project (no. 2015Z115), and the Fundamental Research Funds for the Central Universities (no. 30918014108).

References

[1] L. Qi, Q. He, F. Chen et al., "Finding all you need: web APIs recommendation in web of things through keywords search," *IEEE Transactions on Computational Social Systems*, vol. 6, no. 5, pp. 1063–1072, 2019.

[2] W. Quan, Y. Liu, H. Zhang, and S. Yu, "Enhancing crowd collaborations for software defined vehicular networks," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 80–86, 2017.

[3] L. Qi, R. Wang, C. Hu, S. Li, Q. He, and X. Xu, "Time-aware distributed service recommendation with privacy-preservation," *Information Sciences*, vol. 480, pp. 354–364, 2019.

[4] Y. Zhang, K. Wang, Q. He et al., "Covering-based web service quality prediction via neighborhood-aware matrix factorization," *IEEE Transactions on Services Computing*, 2019.

[5] C. V. Forecast, *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2017–2022 White Paper*, Wiley, Hoboken, NJ, USA, 2019.

[6] F.-H. Tseng, H.-H. Cho, K.-D. Chang, J.-C. Li, and T. K. Shih, "Application-oriented offloading in heterogeneous networks for mobile cloud computing," *Enterprise Information Systems*, vol. 12, no. 4, pp. 398–413, 2018.

[7] Y. Zhang, C. Yin, Q. Wu, Q. He, and H. Zhu, "Location-aware deep collaborative filtering for service recommendation," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–12, 2019.

[8] X. Xu, Y. Chen, X. Zhang, Q. Liu, X. Liu, and L. Qi, "A blockchain-based computation offloading method for edge computing in 5 g networks," *Software: Practice and Experience*, 2019.

[9] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya, "Computation offloading for service workflow in mobile cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 12, pp. 3317–3329, 2014.

[10] T. Shi, M. Yang, X. Li, Q. Lei, and Y. Jiang, "An energy-efficient scheduling scheme for time-constrained tasks in local mobile clouds," *Pervasive and Mobile Computing*, vol. 27, pp. 90–105, 2016.

[11] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *Proceedings of the 2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 352–357, IEEE, Toronto, Canada, 2014.

[12] H. Wu, W. Knottenbelt, and K. Wolter, "An efficient application partitioning algorithm in mobile environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 7, pp. 1464–1480, 2019.

[13] E. Benkhelifa, T. Welsh, L. Tawalbeh, Y. Jararweh, and A. Basalamah, "User profiling for energy optimisation in mobile cloud computing," *Procedia Computer Science*, vol. 52, pp. 1159–1165, 2015.

[14] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *Proceedings of the 35th Annual IEEE International Conference on Computer Communications*, pp. 1–9, IEEE, San Francisco, CA, USA, April 2016.

[15] L. Qi, Y. Chen, Y. Yuan, S. Fu, X. Zhang, and X. Xu, "A Qos-aware virtual machine scheduling method for energy conservation in cloud-based cyber-physical systems," *World Wide Web*, pp. 1–23, 2019.

[16] X. Xu, S. Fu, Q. Cai et al., "Dynamic resource allocation for load balancing in fog environment," *Wireless Communications and Mobile Computing*, vol. 2018, Article ID 6421607, 15 pages, 2018.

[17] E. Ahmed and M. H. Rehmani, *Mobile Edge Computing: Opportunities, Solutions, and Challenges*, Elsevier, Amsterdam, Netherlands, 2017.

[18] H. Peng, W.-S. Wen, M.-L. Tseng, and L.-L. Li, "Joint optimization method for task scheduling time and energy

consumption in mobile cloud computing environment," *Applied Soft Computing*, vol. 80, pp. 534–545, 2019.

[19] K. Zhang, Y. Mao, S. Leng et al., "Energy-efficient offloading for mobile edge computing in 5 g heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.

[20] Y. Sun, F. Lin, and H. Xu, "Multi-objective optimization of resource scheduling in fog computing using an improved NSGA-II," *Wireless Personal Communications*, vol. 102, no. 2, pp. 1369–1385, 2018.

[21] K. Gai, M. Qiu, and H. Zhao, "Energy-aware task assignment for mobile cyber-enabled applications in heterogeneous cloud computing," *Journal of Parallel and Distributed Computing*, vol. 111, pp. 126–135, 2018.

[22] X. Xu, X. Zhang, H. Gao, Y. Xue, L. Qi, and W. Dou, "Become: blockchain-enabled computation offloading for iot in mobile edge computing," *IEEE Transactions on Industrial Informatics*, 2019.

[23] Z. Chen, J. Hu, G. Min, and X. Chen, "Effective data placement for scientific workflows in mobile edge computing using genetic particle swarm optimization," *Concurrency and Computation: Practice and Experience*, Article ID e5413, 2019.

[24] A. Zhu, S. Guo, M. Ma et al., "Computation offloading for workflow in mobile edge computing based on deep Q-learning," in *Proceedings of the 2019 28th Wireless and Optical Communications Conference (WOCC)*, pp. 1–5, IEEE, Beijing, China, May 2019.

[25] S. Wang, Y. Zhao, J. Xu, J. Yuan, and C.-H. Hsu, "Edge server placement in mobile edge computing," *Journal of Parallel and Distributed Computing*, vol. 127, pp. 160–168, 2019.

[26] X. Xu, Y. Xue, L. Qi et al., "An edge computing-enabled computation offloading method with privacy preservation for internet of connected vehicles," *Future Generation Computer Systems*, vol. 96, pp. 89–100, 2019.

[27] Y. Liu, M. J. Lee, and Y. Zheng, "Adaptive multi-resource allocation for cloudlet-based mobile cloud computing system," *IEEE Transactions on Mobile Computing*, vol. 15, no. 10, pp. 2398–2410, 2015.

[28] K. Dolui and S. K. Datta, "Comparison of edge computing implementations: fog computing, cloudlet and mobile edge computing," in *Proceedings of the 2017 Global Internet of Things Summit (GIoTS)*, pp. 1–6, IEEE, Geneva, Switzerland, June 2017.

[29] X. Xu, Q. Liu, Y. Luo et al., "A computation offloading method over big data for iot-enabled cloud-edge computing," *Future Generation Computer Systems*, vol. 95, pp. 522–533, 2019.

[30] W. Quan, N. Cheng, M. Qin, H. Zhang, H. A. Chan, and X. Shen, "Adaptive transmission control for software defined vehicular networks," *IEEE Wireless Communications Letters*, vol. 8, no. 3, pp. 653–656, 2018.

[31] K. Peng, V. C. M. Leung, X. Xu, L. Zheng, J. Wang, and Q. Huang, "A survey on mobile edge computing: focusing on service adoption and provision," *Wireless Communications and Mobile Computing*, vol. 2018, Article ID 8267838, 16 pages, 2018.

[32] X. Xu, C. He, Z. Xu, L. Qi, S. Wan, and M. Z. A. Bhuiyan, "Joint optimization of offloading utility and privacy for edge computing enabled IoT," *IEEE Internet of Things Journal*, 2019.

[33] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "Mobile edge computing: survey and research outlook," 2017, https://arxiv.org/abs/1702.05309.

[34] K. Peng, Y. Zhang, X. Wang, X. Xu, X. Li, and V. C. M. Leung, *Computation Offloading in Mobile Edge Computing*, Springer International Publishing, Cham, Switzerland, 2019.

[35] B. Li, M. He, W. Wu, A. Sangaiah, and G. Jeon, "Computation offloading algorithm for arbitrarily divisible applications in mobile edge computing environments: an OCR case," *Sustainability*, vol. 10, no. 5, p. 1611, 2018.

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

[37] X. Xu, Y. Li, T. Huang et al., "An energy-aware computation offloading method for smart edge computing in wireless metropolitan area networks," *Journal of Network and Computer Applications*, vol. 133, pp. 75–85, 2019.

[38] P. Mach and Z. Becvar, "Mobile edge computing: a survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[39] T. Huang, F. Ruan, S. Xue, L. Qi, and Y. Duan, "Computation offloading for multimedia Workflows with deadline constraints in cloudlet-based mobile cloud," *Wireless Networks*, pp. 1–15, 2019.

[40] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[41] Y. Wang, L. Wu, X. Yuan, X. Liu, and X. Li, "An energy-efficient and deadline-aware task offloading strategy based on channel constraint for mobile cloud workflows," *IEEE Access*, vol. 7, pp. 69858–69872, 2019.

[42] X. Xu, S. Fu, Y. Yuan et al., "Multiobjective computation offloading for workflow management in cloudlet-based mobile cloud using NSGA-II," *Computational Intelligence*, vol. 35, no. 3, pp. 476–495, 2018.

[43] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 16, no. 8, pp. 4924–4938, 2017.

[44] Y. Zhang, G. Cui, S. Deng, F. Chen, Y. Wang, and Q. He, "Efficient query of quality correlation for service composition," *IEEE Transactions on Services Computing*, 2018.

[45] B. Liu, H. Huang, S. Guo, W. Chen, and Z. Zheng, "Joint computation offloading and routing optimization for uav-edge-cloud computing environments," in *Proceedings of the 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing*, pp. 1745–1752, IEEE, Guangzhou, China, October 2018.

[46] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, Fog et al.: a survey and analysis of security threats and challenges," *Future Generation Computer Systems*, vol. 78, pp. 680–698, 2018.

[47] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang, "The case for cyber foraging," in *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, pp. 87–92, ACM, Copenhaguen, Denmark, July 2002.

[48] K. Peng, M. Zhu, Y. Zhang et al., "An energy- and cost-aware computation offloading method for workflow applications in mobile edge computing," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, 2019.

[49] W. Shi, F. Liu, H. Sun, and Q. Pei, *Edge Computing*, Science Press, Beijing, China, 2018.

[50] J. Vilaplana, F. Solsona, I. Teixidó, J. Mateo, F. Abella, and J. Rius, "A queuing theory model for cloud computing," *The Journal of Supercomputing*, vol. 69, no. 1, pp. 492–507, 2014.