Hindawi
Mobile Information Systems
Volume 2021, Article ID 5578465, 11 pages
https://doi.org/10.1155/2021/5578465

*Research Article*

# Efficient Computation Offloading for Service Workflow of Mobile Applications in Mobile Edge Computing

**Youwei Yuan,[1,2] Lu Qian ,[1,2] Gangyong Jia,[1] Longxuan Yu,[3] Zixuan Yu,[3] and Qi Zhao[1]**

[1]*School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China*
[2]*Key Laboratory of Brain Machine Collaborative Intelligence of Zhejiang Province, Hangzhou 310018, China*
[3]*School of Digital Media and Design, Hangzhou Dianzi University, Hangzhou 310018, China*

Correspondence should be addressed to Lu Qian; qianlu@hdu.edu.cn

Received 23 February 2021; Revised 17 March 2021; Accepted 26 March 2021; Published 5 April 2021

Academic Editor: Xiaoxian Yang



Edge computing has become a promising solution to overcome the user equipment (UE) constraints such as low computing capacity and limited energy. A key edge computing challenge is providing computing services with low service congestion and low latency, but the computing resources of edge servers were limited. User task randomness and network inhomogeneity brought considerable challenges to limited-resource MEC systems. To solve these problems, the presented paper proposed a blocking- and delay-aware schedule strategy for MEC environment service workflow offloading. First, the workflow was modeled in mobile applications and the buffer queue in servers. Then, the server collaboration area was divided through a collaboration area division method based on clustering. Finally, an improved particle swarm optimization scheduling method was utilized to solve this NP-hard problem. Many simulation results verified the effectiveness of the proposed scheme. This method was superior to existing methods, which effectively reduces the blocking probability and execution delay and ensures the quality of the experience of the user.

## 1. Introduction

In recent years, the rapid growth of mobile Internet and the proliferation of user devices [1, 2] has led to more new mobile applications such as face recognition, augmented reality [3], and mobile online games emerging. These services require more computation resources due to intensive computation. However, mobile devices lack computing resources, memory, and battery capacity. These applications often exceed the computing power of ordinary mobile devices. Mobile devices may not be able to complete computationally intensive tasks within the current latency limit. However, cloud computing has rescued mobile applications, to a certain extent, via high resource requirements, while providing computing power and simple centralized architecture that helps run these applications with influential economies of scale. However, the 5 G era with ultra-high bandwidth and ultra-low latency [4] means that cloud computing often fails to meet the strict delay-sensitive application requirements due to unpredictable network delays and expensive bandwidth [5]. To remedy these limitations, the utilization of computing resources at the network edge has been proposed as a solution, and mobile edge computing has recently been utilized as a new computing paradigm. Designed to reduce latency and transfer computing power to the network edge, mobile edge computing can effectively mitigate the task transmission delays due to its user proximity, and it can provide effective computing services. Being a promising solution in a mobile edge computing system, users offload tasks requiring a large number of computing resources in mobile devices to the server connected to the base station edge for execution [6], and wireless devices with limited battery capacity can ease some tasks via MEC, which can significantly reduce delays and extend battery life [7]. The Internet of things is also a very important solution in the edge environment. The concept of smart city is given in [8].

Aiming at task offloading with delay constraints in the mobile edge environment, an optimization algorithm for minimizing delay was proposed, which offloaded tasks with strict delay boundaries to the edge cloud and offloaded tasks with loose delay boundaries to the remote cloud [9]. Task allocation in the Internet of things is solved by offloading computing intensive tasks to the cloud server so as to achieve real-time monitoring [10]. For the task offloading problem of multiuser shared resources in the mobile edge environment, a corresponding joint optimization problem optimization algorithm was developed to minimize the user task completion time [11]. Some authors utilized the software-defined network to express the workflow offloading problem as an NP-hard problem and proposed a workflow task offloading scheme, which effectively reduced the task execution time [12]. The problem of task offloading has been studied for multidevice systems [13]. Multiuser single-server systems have been studied in [14–18].

Through those documents, each server was found to run independently in these schemes. However, because user requests arrived at the server randomly and frequently, the problem of load imbalance was important [19]. Users in areas with heavy loads will inevitably have a bad experience. This was a massive challenge for MECs with limited communication and computing resources. Besides, service workflow and network state are also important difficulties in the edge computing environment. A dynamic reconfiguration scheme of service workflow in mobile e-commerce environment based on cloud edge was proposed, which was more suitable for cloud edge environment [20]. A training resource allocation strategy based on reinforcement learning was proposed [21]. The strategy dynamically generates an appropriate resource allocation scheme according to the system state so as to maximize the trusted gain of the service. A sparsity alleviation recommendation approach was presented in [22], which achieves a better recommendation performance for user to choose better edge servers by the model. An attempt to employ neural network technique for QoS prediction is shown in [23], and the network-based time series predicts the periodic trend of user behavior in [24]. A VANET routing decision scheme based on the Manhattan mobility model was proposed to get better network scheduling [25]. When the edge servers form clusters, the quality of user QoS is still an important issue. A load balancing scheme to minimize the blocking probability and task waiting time for two servers was presented [26].

A promising solution is cooperating between MECs [27]. With the trend of deploying edge servers, each server usually has some neighbors nearby. Simultaneously, all servers rarely overload. An intelligent monitoring system is proposed to control the load state of each edge server [28]. The MEC cluster can balance workloads among geographically distributed servers by offloading servers with large computing workloads to neighboring servers with small computing workloads and coordinate between them to serve mobile users. To meet user needs, maintaining a load balance between servers is an important issue. Task offloading between MECs is not straightforward, and two major challenges are clear as follows:

(1) The problem of MEC cooperation collaboration area division is as follows: edge servers with small computing workloads need to help servers with large computing workloads; however, it is unrealistic for two servers to share a large delay to cooperate, which will cause execution delay. Dividing the collaboration area first is necessary. On the one hand, the collaborative regions can capture the MEC inhomogeneity and speed up network stabilization [29]. On the other hand, effective collaboration area division helps reduce the search space of the strategy set.

(2) Task scheduling problems in MEC cooperation are as follows: This problem is NP-hard [30]. The goal to find an optimal approximate solution must be obtained in a very short time.

To meet these challenges, a blocking- and delay-aware strategy for service workflow offloading in the MEC environment was proposed. First, the service workflow in mobile applications and buffer size in edge serves were defined. Then, the computation offloading problem was formulated while considering the blocking probability and execution delay. Finally, a collaboration area division method and an improved immune particle swarm optimization scheduling algorithm were presented to solve this problem. Many simulation results verified the effectiveness of the proposed scheme.

The rest of the presented paper was structured as follows. Section 2 presents the system model of the MEC network. Section 3 presents an optimization problem and introduces the solution and optimization algorithm of the problem. Section 4 validates the model against simulation results. Section 5 concludes the paper and identifies future directions.

## 2. System Model and Problem Formulation

In this section, a workflow offload framework in the MEC environment is introduced. Figure 1 shows that the MEC system consisted of the user device and edge servers equipped with base stations. User device includes mobile phones and laptops. A user device executed an application, and the application was modeled as a workflow. If the workflow required a lot of calculation, it would be offloaded to the edge server through the base station. The workflow arriving at the server is decomposed into tasks. This server would execute the workflow locally or redirect it to other servers according to its load status. Tasks enter the server buffer queue according to the principle of first come first served, or are dispatched by the server to other servers. The tasks scheduled to other servers are transmitted through the wired network between the base stations and enter the task buffer according to the principle of first come first service. When the server completes the task, the desired task results and the required parameters will be returned to the user's device. The MEC servers cooperated to complete tasks to meet user delay and energy consumption requirements. Also, the system time was divided into several identical time slots $\tau = \{1, 2, \ldots, n\}$.

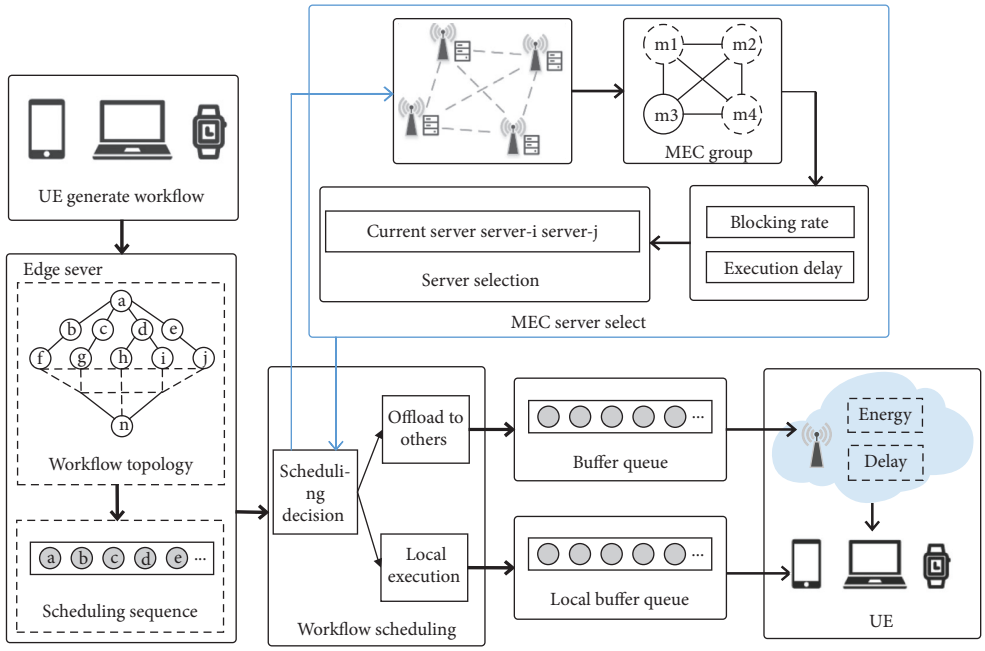


Figure 1: The framework of workflow offload in the MEC system.

*2.1. Service Workflow Model.* In the MEC system, the user device executed the application, and the application was modeled as a workflow. The workflow user-submitted was defined as $W = \{T, E\}$, where $T = \{t_1, t_2, t_3, \ldots, t_n\}$ represented the task set in server-$l$, $t_i = \{u_i, v_i\}$, $u_i$ represented the task size, and $v_i$ was the processing density required for $t_i$ which represented the dependencies between tasks, and the tasks with dependencies needed to be executed in order.

The task execution queue for the workflow was generated and then the workflow entered the edge server buffer queue. A single task flow was defined as the smallest unit in workflow scheduling. There were dependencies between tasks in the workflow and transmission results between different servers that would cause unnecessary overhead.

*2.2. MEC Edge Server Model.* The MEC system consisting of S servers that form a server cluster and $N$ users were considered. Each server had one base station (BS) that was connected to users via wireless cellular links. This was a highly collaborative edge computing system, and every server was connected via a wired network. There was a buffer queue in every server to store the requests from local users. Each buffer size represents the ability of a server to accept tasks. To simplify the problem, the size of the buffer queue in each server was assumed to be the same. Workflow service was based on the first-come-first-served policy (FCFS), and workflows that have not yet been executed would wait in the buffer queue. Then, the server will continue to accept tasks. In fact, the server capacity is limited. When a workflow was offloaded to the MEC, it first entered the task buffer queue; then, the MEC server provided computing resources for it. It would be deleted from the buffer queue after the workflow was completed. The workflow would be executed directly if the CPU was idle; if the CPU was busy, the requested would store in the buffer queue. The task buffer queue has a maximum capacity $Q_j^{\max}$. $Q_j(\tau)$ presented the queue backlog for unprocessed workflows from server-$j$ at slot $\tau$:

$$Q_j(\tau) = \{w_1, w_2, w_3, \ldots, w_n\}, \quad (1)$$

where $w_i$ represented the $i$-th waiting workflow in the buffer queue.

The load status of the server $i$ was defined as $L_i(\tau) = Q_j(\tau)/Q_j^{\max}$ at slot $\tau$; if $L_i(\tau) \geq$ threshold, this was considered a server with large computing workloads and was described as a hot server. Otherwise, it was defined as a nonhot server. In urban areas, some nonhot MEC servers are located near the high ones [31]. In the present scheme, only when the buffer exceeded server's threshold would the server offload its workflow to other servers. The goal of the presented paper was to schedule these workflows that exceeded the threshold. The server will not schedule the workflow outward if the load state of each server does not exceed the threshold because the waiting time of the task is acceptable, which will cause unnecessary overhead. When the load state exceeds the threshold, the server will schedule the workflow outward to achieve a smaller task waiting delay and reduce the possibility of blocking.

When a large number of workflows were offloaded to the server in a certain period, the task buffer queue of the server would continue to grow. The update process of the buffer queue at slot $\tau$ was as follows:

$$Q_i(\tau+1) = \min\left\{\max\left\{Q_i(\tau) - \sum_{j=0,j\neq i}^{j\in R} x_{i,j}(\tau) - B_i(\tau), 0\right\} + \sum_{j=0,j\neq i}^{j\in R} x_{j,i}(\tau) - A_i(\tau), Q_i^{\max}\right\}, \tag{2}$$

where $A_i(\tau)$ represented the number of workflows offloaded via users locally at slot $\tau$, $B_i(\tau)$ represented the number of workflows handled by the MEC server-$i$, and $x_{i,j}(\tau)$ represented the number of workflows allocated from server-$i$ to server-$j$ at slot $\tau$. When the buffer queue was full and workflows were still arriving, the workflows would be blocked and fail to offload. The user failing to offload would seriously affect the user quality of experience (QoE).

Besides, the blocking probability $P_f$ was defined as follows:

$$P_f(\tau) = \frac{N_f(\tau)}{N_{\text{all}}(\tau)}, \tag{3}$$

where $N_f(\tau)$ represented the number of workflows that failed to offload and $N_{\text{all}}(\tau)$ represented the number of workflows.

*2.3. Computation Delay.* In this model, the execution delay represented from when the workflow arrived at the server to when it finished. It mainly concluded transmission delay, waiting delay, computing delay, and extra time due to blocking, while ignoring the time to download the calculation result from the server. The basic principle was that, when compared with the workflow before the calculation, the size of the calculation result was usually smaller.

Assuming a workflow $w_s$ generated at slot $\tau$, the transmission time of the workflow from server-$i$ to server-$j$ was as follows:

$$T_{i,j}^{\text{trans}}(w_s) = \frac{\sum_{l=1}^{n} u_l}{BW_{i,j}}, \tag{4}$$

where $u_l$ represented the size of the task in $w_s$ and $BW_{i,j}$ was the bandwidth between server-$i$ and server-$j$.

The workflow waiting time at server-$i$ was as follows:

$$T_i^{\text{wait}}(\tau) = \sum_{j=1}^{m} T_i^{\text{exec}}\left(w_j\right), \tag{5}$$

where $m$ represented the number of workflows in the buffer queue.

The computing time of $t_k$ which in $w_s$ was as follows:

$$T_i^{\text{exec}}(t_k) = \frac{v_k}{f_{i,k}(\tau)}, \tag{6}$$

where $f_{i,k}$ indicated the calculation frequency assigned to $t_k$ by server-$i$. The MEC server was deployed to provide faster computing power for the UE.

The cost was zero if the UE offloaded the workflow $w_s$ successfully. If it failed, it would continue to upload within the UE's acceptable time $w$. The extra time caused by offload failed $T_s^{\text{fail}}$ was as follows:

$$T_s^{\text{fail}} = \begin{cases} 0, & \text{success} \\ f_1, i \in \{0,1,\ldots\}, & \text{fail} \end{cases} \tag{7}$$

The execution delay of the workflow $w_s$ was as follows:

$$T(w_s) = \begin{cases} T_i^{\text{wait}}(w_s) + T_i^{\text{fail}}(w_s) + T_i^{\text{exec}}(w_s), & \text{server} - i \\ T_{i,j}^{\text{trans}}(w_s) + T_j^{\text{wait}}(w_s) + T_j^{\text{fail}}(w_s) + T_j^{\text{exec}}(w_s), & \text{server} - j \end{cases} \tag{8}$$

*2.4. Problem Formulation.* The optimization goal of the presented paper was to reduce execution delay while ensuring a low task blocking possibility. The offloading failure cost was lower QoS and additional overhead caused by task retransmission, replication, or scheduling. Through the above analysis, this paper involved a joint blocking probability and task delay algorithm, while meeting a low blocking probability, thereby reducing execution delay as much as possible. The optimization model was as follows:

$$\begin{aligned} &\min \quad \alpha P^{\text{fail}}(\tau) + \frac{1}{m}\sum_{i=1}^{m} T(w_i), \\ &s.t. \quad 0 \le Q_i(\tau) \le Q_i^{\max}, \quad \forall i \in S, \\ &f_{i,k} \in \left[f_i^{\min}, f_i^{\max}\right], \quad \forall i \in S, \\ &0 \le T_i^{fail} \le w, \quad \forall i \in S. \end{aligned} \tag{9}$$

Equation (9) was the objective function, where $m$ represented the number of workflows that need to be coordinated and $\alpha$ was the penalty coefficient for blocking. Equation (9) ensured that the buffer queue would not exceed the maximum length. Equation (9) was a constraint to allocate computing resources to users. Equation (9) gave the time limitation to task failure.

## 3. The Proposed Scheme

In this section, a blocking- and delay-aware schedule strategy for service workflow offloading in the MEC environment was proposed. The scheme included a collaboration area division method and an improved particle swarm algorithm. Figure 2 shows the flow of each time slice. First, the hot server sent the cooperation request, and then the cooperation area was divided by Algorithm 1. Each collaboration area randomly selected a server as the leader. The leader generated workflow scheduling decisions via Algorithm 2.

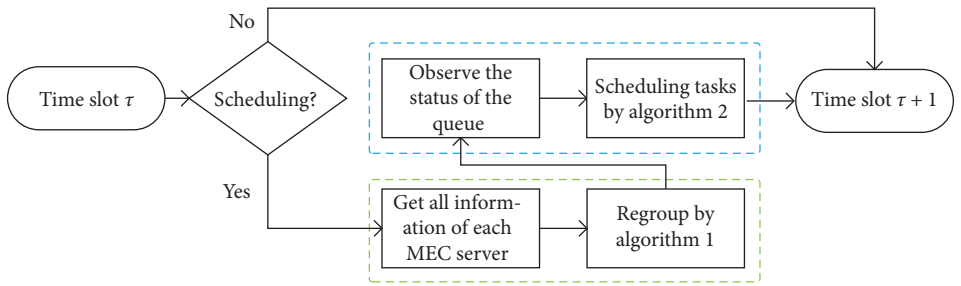


Figure 2: The process of the scheme.

Input: $Q_i$, $BW_{i,j}$, $w$, $f_{i,j}$, $B$, $t$, $epoch$, $i$, $j \in R$
Output: $Gbest$
(1) for $i = 1,2, \ldots, s$ do
(2) $C_i = \{m_i\}$
(3) end for
(4) $q = s$
(5) while $q > k$ do
(6) for $i = 1$ to $q$ do
(7) for $j = i + 1$ to $q$ do
(8) Calculate $R(C_i, C_j)$ by (10)
(9) get the nearest cluster $C_{i^*}$ and $C_{j^*}$
(10) end for
(11) end for
(12) for $j = j^* + 1, j^* + 2, \ldots, q$ do end for
(13) Renumber $C_j = C_{j-1}$
(14) end for
(15) $q = q - 1$
(16) end while

Algorithm 1: A collaboration area division method based on clustering.

Input: $Q_i$, $BW_{i,j}$, $w$, $f_{i,j}$, $B$, $t$, $epoch$, $i$, $j \in R$
Output: $Gbest$
(1) Set $count$, $Gk$, $c_1$, $c_2$, $w_i$, $w_e$, $epoch$, $seed$, $X$, $V$
(2) for $l = 1$ to $seed$ do
(3) initialize velocity $v_{i,j}$ and position $x_{i,j}$ for particle
(4) evaluate particle $X_l$ and set $pbest_l = X_l$
(5) end for
(6) for $k = 1$ to $Gk$ do
(7) update $w$ by (10);
(8) for $i = 1$ to $seed + s$ do
(9) for $j = 1$ to $h$ do
(10) update $X$ and $V$ by (10)
(11) Constrain the particle boundary
(12) end for end for
(13) if $\text{fit}(X_i^k) < \text{fit}(pbest_i)$ then
(14) $pbest_i = X_i^k$
(15) if $\text{fit}(X_i^k) < \text{fit}(Gbest)$ then
(16) $Gbest = X_i^k$
(17) end for
(18) immune operation through formulas (14)–(16)
(19) end for

Algorithm 2: An immune particle swarm algorithm for computing offloading in MEC environment.

*3.1. A Collaboration Area Division Method Based on Clustering.* Collaboration area division was necessary because it could improve resource utilization. The MEC server would be grouped according to the server load status. There was a requirement in the collaboration area division: proximity principle; that is, the overload of MEC servers could only be assisted by adjacent server with high transmission rate between them.

The distance between the two clusters was as follows:

$$R\left(C_i, C_j\right) = \frac{1}{mn}\sum_{l=1}^{m}\sum_{k=1}^{n}\log_2\left(BW_{k,l}+1\right), \tag{10}$$

where $m$ and $n$ represented the number of servers in $C_i$ and $C_j$ and $BW_{k,l}$ represented the bandwidth between server-$k$ and server-$l$.

The proposed Algorithm 1 started with initialization (lines 1-2), in which $s$ represented the number of servers. $q$ was defined as the number of clusters (line 4). If the value of $q$ is higher than $k$, the two nearest clusters would be searched and merged. Some clusters would be renumbered (line13-14), and $q$ would be updated (line 16). Finally, the MEC servers were divided into $k$ clusters.

*3.2. An Improved Particle Swarm Optimization Algorithm Framework.* Particle swarm optimization (PSO) is a reliable algorithm to obtain feasible solutions from a large search space by using the principle of evolution. The problem of minimizing blocking probability and execution delay is NP hard. The goal of the problem proposed in this paper is to find an optimal approximate solution. To solve this problem, an immune particle swarm optimization-based algorithm (IPSOA) is proposed.

*3.2.1. Particle Encode and Particle Fitness Function.* In the collaborative offloading problem, the collaboration will be engaged in tasks that exceed the server load threshold. The workflows that needed to be scheduled at slot $\tau$ were as follows:

$$W(\tau) = \{w_1, w_2, w_3, ..., w_h\}. \tag{11}$$

The solution was defined as a particle swarm $X = \{X_1, X_2, \ldots, X_{\text{seed}}\}$, in which *seed* represented the amount of particle swarm. Each particle $X_i$ could be represented as a n-dimensional tuples $\{x_{i,1}, x_{i,2}, x_{i,3}, \ldots, x_{i,h}\}$. The relationship between the workflow and the server was defined as mapping relations. And $x_{i,j} = 5$ represented that the $j$-th workflow would be redirected to edge server-5, and $i$ was the number of the particle swarm. Each particle was associated with a position $x_{i,j}$ and a velocity $v_{i,jj}$. The value of $x_{i,j}$ and $v_{i,jj}$ was restricted within intervals $[x_{\min}, x_{\max}]$ and $[v_{\min}, v_{\max}]$. The random initialization method was used to generate the initial particle. Every particle was considered as a solution in the present scheme. $G_{\text{best}}$ and $P_{\text{best}}$ were defined as the global best particle and the personal best particle, respectively.

The fitness function was an essential basis for measuring the particle positions. In the present article, the blocking probability and execution delay of the workflow that exceeded the server load threshold as the value of fitness calculation were utilized. The calculation method of particle fitness was as follows:

$$\text{fit}(X) = \alpha P_f(X) + T(X), \tag{12}$$

where $X$ was the particle matrix. $T(X)$ was the execution delay of the particle, $P_f(X)$ represented the particle blocking probability, and $\alpha$ was the penalty coefficient for blocking.

*3.2.2. Particle Update.* In the IPSO framework, every particle moved towards to $G_{\text{best}}$ and $P_{\text{best}}$. The velocity $v_{i,j}$ and the position $x_{i,j}$ of particle $X$ were updated by equation (13). $w$ was the inertia weight, and it would be updated by equation (13), and the value of the inertia weight $w$ was restricted within interval $[w_i, w_e]$. $G_k$ represented the maximum number of iterations. $r_1$ and $r_2$ were random numbers distributed within the interval [0,1]. $c_1$ and $c_2$ were the individual cognition component and the social communication component, respectively.

$$\begin{aligned} v_{i,j}^{k+1} &= wv_{i,j}^{k} + c_1 r_1\left(pbest_i - x_{i,j}^{k}\right) + c_2 r_2\left(Gbest - x_{i,j}^{k}\right), \\ x_{i,j}^{k+1} &= x_{i,j}^{k} + v_{i,j}^{k+1}, \\ w &= (w_i - w_e)\cdot\frac{(G_k - k)}{(G_k + w_e)}. \end{aligned} \tag{13}$$

*3.2.3. The Immune Strategy of Particle Swarm Optimization.* In the standard particle swarm algorithm, an immune operation was added, and the immune operation of the immune algorithm helped the particles to easily get rid of the optimal local state, thereby ensuring the global nature of the solution of the workflow scheduling scheme. In the immune algorithm, the system needed to first calculate the antibody concentration $\text{Conc}(X_i^k)$ as follows:

$$\text{Conc}\left(X_i^k\right) = \frac{1}{h+s}\sum_{j=1}^{h+s} sr\left(X_i^k, X_j^k\right), \tag{14}$$

where $h + s$ was the original particle plus the size of the newly generated antibody and $sr(X_i^k, X_j^k)$ was the result of particle similarity determination:

$$\begin{aligned} sr\left(X_i^k, X_j^k\right) &= \begin{cases} 1, & \text{aff}\left(X_i^k, X_j^k\right) < \varepsilon \\ 0, & \text{aff}\left(X_i^k, X_j^k\right) \geq \varepsilon \end{cases}, \\ \text{aff}\left(X_i^k, X_j^k\right) &= \sqrt{\sum_{d=1}^{n}\left(X_{i,d}^k - X_{j,d}^k\right)^2}, \end{aligned} \tag{15}$$

where $\text{aff}(X_i^k, X_j^k)$ represented the affinity between antibodies and $\varepsilon$ was the threshold utilized to determine the similarity between particles, and then the incentive $\text{sim}(X^k)$ was calculated as follows:

$$\text{sim}\left(X^k\right) = \text{fit}\left(X^k\right)\cdot\ \exp\left(-\beta\cdot\text{Conc}\left(X^k\right)\right). \tag{16}$$

Finally, the antibodies were screened and arranged in descending order according to the excitation degree, and the

first $s$ antibodies were selected to enter the next population iteration to maintain the concentration of antibodies within the optimal response range.

*3.2.4. An Immune Particle Swarm Optimization Algorithm for Computing Offloading.* The proposed Algorithm 2 started with an initialization procedure (lines 2-3), in which seed represented the number of particle swarms. For each iteration, $w$ would be updated first (line 7). Then, the position $x_{i,j}$ and the velocity $v_{i,j}$ of the particle were updated (line 10). In each iteration, personal best particle and the global best particle would be updated if a better solution was found (lines 12–15). An immune operation is added (line 16). The complete algorithmic procedure of IPSO was given in Algorithm 2.

## 4. Experiment

In this section, experiments are implemented to verify purposes. Our experiments are implemented on a workstation with an AMD Core CPU and 16GB RAM. The operation system of the workstation is Windows10 Professional, and the Python 3.7 programming language was used. In addition, the fundamental packages NumPy and Tensorflow were used in our experiments, and our experimental data are obtained by simulation.

*4.1. Experimental Setup.* A resource-limited MEC system with 50 servers was considered. The processing capacity of each server was a Poisson distribution that obeyed $\lambda = 0.9$. The size of the task in the workflow was assumed to have obeyed the Poisson distribution $\mu = 0.3$, and the processing density which the task required obeyed the negative exponential distribution. In the model, each server was assumed to have the same configuration and the same size buffer queue.

In this paper, three schemes were utilized as benchmarks.

(1) *No Cooperation Offloading (NCO).* Tasks would be queued for execution at the local server, and no cooperation existed among the MEC servers [32].

(2) *Random Workflow Offloading (RWO).* The workload was offloaded via the MEC, and the MEC servers cooperated. When the buffer of a server reached the threshold, it randomly selected the MEC server to handle the new task [33].

(3) *Greedy Workflow Offloading (GWO).* When the buffer of a server reached the threshold, it would offload to the lowest load MEC server [34].

These three schemes were chosen for study because they were representative schemes. Specifically, scheme 1 prohibited resource sharing. MEC servers collaborated with the other MEC server randomly when the MEC server was overloaded in scheme 2. In scheme 3, it allowed for complete sharing of resources, but the execution time was not considered.

Three main performance indicators were considered to evaluate this scheme: blocking probability of service workflow, the execution delay, and energy consumption.

*4.2. Performance Evaluation.* To better simulate the scheme and show the MEC cooperation area division effect, a comparative experiment was conducted for each group of experiments. Each group of experiment (1) showed the experimental results without the collaboration area and (2) showed the experimental results with the collaboration area.

Figure 3 analyzes the buffer size impact of the MEC server on the blocking probability. The present scheme achieved a lower blocking probability than others. This was because the present scheme accounted for the situation of multiple servers in the collaboration area, and hot servers to offload workflows to a nonhot server compared with schemes 2 and 3 would not occur, so that a lower block probability could be obtained. After the collaboration area division in Figure 3(b), the blocking probability achieved by our approach converged with the blocking probability obtained via scheme 3. The reason was that when the buffer size was small, a few solutions existed for the edge server to offload tasks as the buffer size was too small to accept all tasks. When the buffer size became larger, as expected, it could effectively reduce the blocking probability for the edge server, which had more space to receive service requests. After the collaboration area was divided, schemes 2 and 3 had a better solution set space.

The average time delay versus different buffer sizes for the four different schemes is shown in Figure 4. The blocking probability of schemes 1, 2, and 3 was significantly higher than that of the present scheme. The higher offloading failure probability of scheme 1 was predictable. Because it was not shared, an increase in the number of tasks would cause tasks to accumulate in the buffer. Both the greedy scheme and the proposed scheme had lower failure rates. The reason was that those schemes were considered to be cooperative with nonhot edge servers. For all schemes, the average time delay increased with the buffer size. This result was because the request of the users was more likely to be stored in the server if the buffer capacity became larger. Moreover, the length of the task queue was longer. After grouping, schemes 2, 3, and the present had a delay reduction; when the collaboration area was divided, the delay between servers in the same area was small, which once again illustrated the effectiveness of the present collaboration area division method.

Figure 5 depicts the blocking probability versus threshold in the same buffer size. Figure 5(a) shows that the blocking probability decreased as the threshold increased. Scheme 1 had the highest block probability as it did not cooperate with other servers; the scheme 2 and 3 reduced the blocking probability to a certain extent but not as well as the present scheme. The present scheme could get reduction blocking probability. After the cooperation area was divided, the blocking probability of schemes 2 and 3 had been reduced. Our scheme always obtains a low blocking probability.

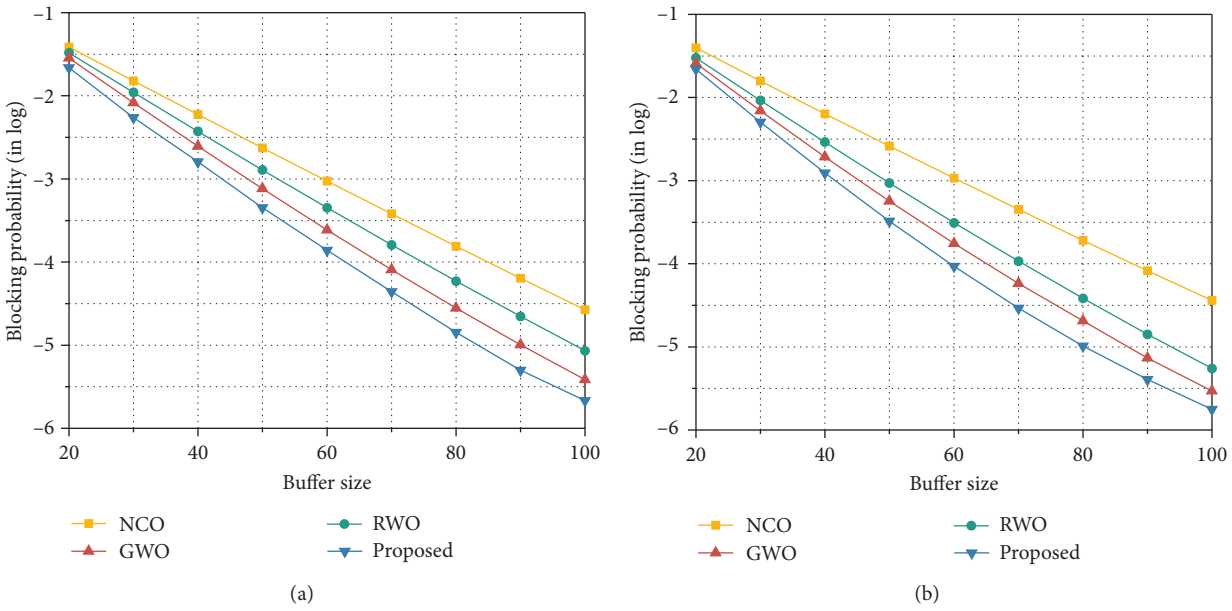


Figure 3: Block probability versus buffer size.

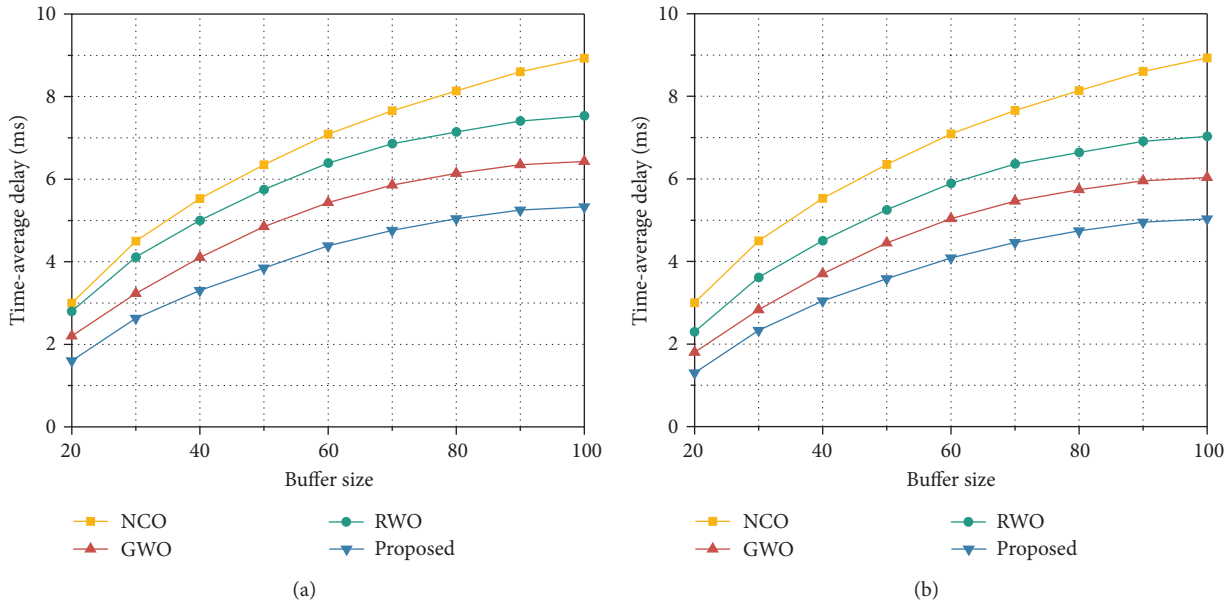


Figure 4: Time-average delay versus buffer size.

To compare the energy consumption, the energy consumption of the four schemes was simulated and calculated. Figure 6 demonstrates the impact of the number of workflows on energy consumption in different schemes. Figure 6 also shows that the task offloading energy consumption increased as the number of workflows increased. This was because more energy was the cost to compute and schedule workflows. In comparison, the present scheme could maintain a lower overhead than NCO, RWO, and GWO as it found a better solution in the iterative process.

To examine the performance of the IPSO algorithm and compare it with the standard PSO algorithm, an example calculation was conducted, and the fitness values of the particles under the same workflow and the number of particle swarms were compared. Figure 7 shows the results from iterations. Fix($X$) represents the fitness

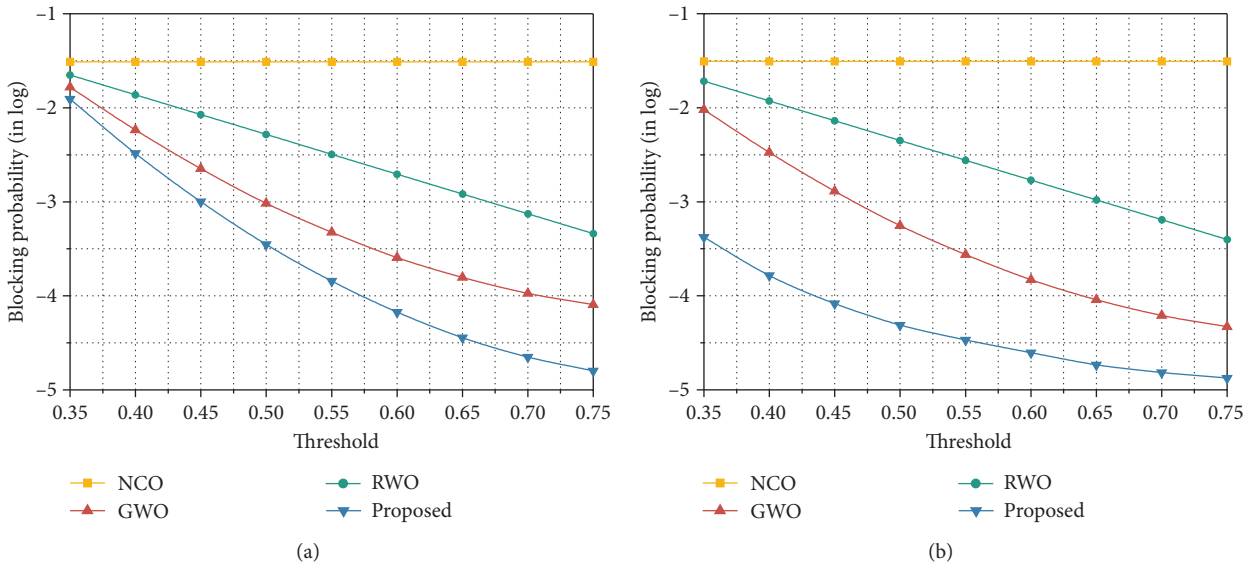


Figure 5: Block probability versus threshold.

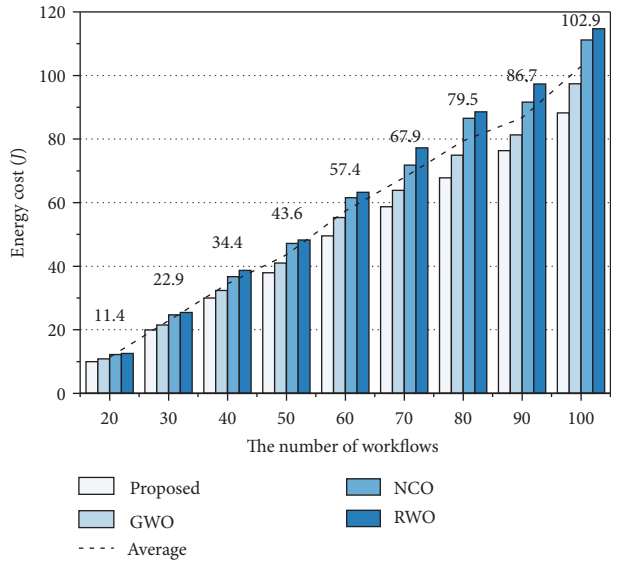


Figure 6: Energy cost versus the number of workflows.

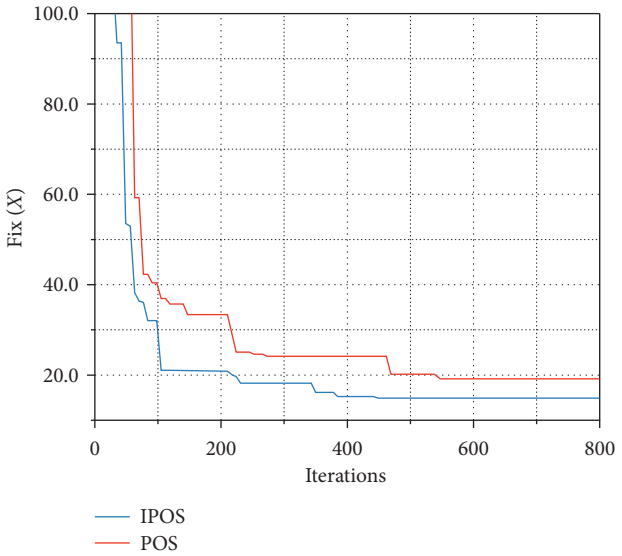


Figure 7: Fix($X$) versus the iterations.

values of particles. The IPSO could converge to the global optimal almost every time, and the convergence speed was faster, while the standard PSO algorithm could search for the optimal result within about 500 iterations, while the IPSO algorithm was about 400. Compared with the standard PSO algorithm, the IPSO fell into the local optimum less. This was because an immune operation was added to the improved particle swarm algorithm to help the particles get rid of the local optimal state.

## 5. Conclusions

Efficient offloading of service workflow in mobile applications was an essential content in edge computing research. In this paper, the workflow buffer queue and workflow execution delay were modeled. A cluster-based division of collaborative areas was designed, and an immune particle swarm optimization algorithm combined with service workflow was proposed. In the task scheduling, task blocking and delay are fully considered and combined with the IPSO algorithm. The algorithm was integrated into the immune algorithm in the particle swarm algorithm, which solved the defect that particles were easy to fall into local optimality and ensured the solution's globality. Simulation experiment results showed that the IPSO algorithm could effectively reduce execution delay and blocking probability.

## Data Availability

The data and python code for the simulations are available from the corresponding authors upon request.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Acknowledgments

The work was supported by the Natural Science Foundation of Zhejiang Province under grant nos. LGG21F010005 and LY19F020044), the National Natural Science Foundation of China under grant no. U20A20386, the Zhejiang Key Research and Development Program under grant no. 2020C01050, and the Key Laboratory Fund General Project under grant no. 6142110190406.

## References

[1] A. Puder and I. Yoon, "Smartphone cross-compilation framework for multiplayer online games," in *Proceedings of the 2010 Second International Conference on Mobile, Hybrid, and On-Line Learning*, pp. 87–92, Sint Maarten, February 2010.

[2] W. Jiang, J. Wu, F. Li, G. Wang, and H. Zheng, "Trust evaluation in online social networks using generalized network flow," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 952–963, 2016.

[3] A. Qin, C. Cai, Q. Wang, Y. Ni, and H. Zhu, "Game theoretical multi-user computation offloading for mobile-edge cloud computing," in *Proceedings of the 2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pp. 328–332, San Jose, CA, USA, March 2019.

[4] T. Taleb, A. Ksentini, and R. Jantti, ""Anything as a service" for 5G mobile systems," *IEEE Network*, vol. 30, no. 6, pp. 84–91, 2016.

[5] H. Gao, L. Kuang, Y. Yin, B. Guo, and K. Dou, "Mining consuming behaviors with temporal evolution for personalized recommendation in mobile marketing apps," *Mobile Networks and Applications*, vol. 25, no. 4, pp. 1233–1248, 2020.

[6] O. Osanaiye, S. Chen, Z. Yan, R. Lu, K.-K. R. Choo, and M. Dlodlo, "From cloud to fog computing: a review and a conceptual live VM migration framework," *IEEE Access*, vol. 5, pp. 8284–8300, 2017.

[7] 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.

[8] G. Jia, G. Han, H. Rao, and L. Shu, "Edge computing-based intelligent manhole cover management system for smart cities," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1648–1656, 2018.

[9] T. Zhao, S. Zhou, X. Guo, and Z. Niu, ". Tasks scheduling and resource allocation in heterogeneous cloud for delay-bounded mobile edge computing," in *Proceedings of the 2017 IEEE International Conference on Communications (ICC)*, pp. 1–7, Paris, France, May 2017.

[10] G. Jia, G. Han, J. Du, and S. Chan, "PMS: intelligent pollution monitoring system based on the industrial Internet of things for a healthier city," *IEEE Network*, vol. 33, no. 5, pp. 34–40, 2019.

[11] H. Q. Le, H. Al-Shatri, and A. Klein, "Efficient resource allocation in mobile-edge computation offloading: completion time minimization," in *Proceedings of the 2017 IEEE International Symposium on Information Theory (ISIT)*, pp. 2513–2517, Aachen, Germany, June 2017.

[12] 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.

[13] G. Jia, G. Han, A. Li, and J. Du, "SSL: smart street lamp based on fog computing for smarter cities," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4995–5004, 2018.

[14] Y. Mao, J. Zhang, S. H. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 16, no. 9, pp. 5994–6009, 2017.

[15] C. You, K. Huang, H. Chae, and B. H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 16, no. 3, pp. 1397–1411, 2017.

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

[17] S. Bi and Y. J. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, no. 6, pp. 4177–4190, 2018.

[18] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, 2015.

[19] R. Beraldi, A. Mtibaa, and H. Alnuweiri, "Cooperative load balancing scheme for edge computing resources," in *Proceedings of the 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 94–100, Valencia, Spain, May 2017.

[20] H. Gao, W. Huang, and Y. Duan, "The cloud-edge-based dynamic reconfiguration to service workflow for mobile ecommerce environments," *ACM Transactions on Internet Technology*, vol. 21, no. 1, p. 1, 2021.

[21] S. Deng, Z. Xiang, P. Zhao et al., "Dynamical resource allocation in edge for trustable internet-of-things systems: a reinforcement learning method," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 6103–6113, 2020.

[22] X. Yang, S. Zhou, and M. Cao, "An approach to alleviate the sparsity problem of hybrid collaborative filtering based recommendations: the product-attribute perspective from user reviews," *Mobile Networks and Applications*, vol. 25, no. 2, pp. 376–390, 2020.

[23] Y. Yin, Z. Cao, Y. Xu, H. Gao, R. Li, and Z. Mai, "QoS prediction for service recommendation with features learning in mobile edge computing environment," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 4, pp. 1136–1145, 2020.

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

[25] H. Gao, C. Liu, Y. Li, and X. Yang, "V2VR: reliable hybrid-network-oriented V2V data transmission and routing considering RSUs and connectivity probability," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–14, 2020.

[26] J. Oueis, E. C. Strinati, and S. Barbarossa, "The fog balancing: load distribution for small cell cloud computing," in *Proceedings of the 2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*, pp. 1–6, Glasgow, UK, May 2015.

[27] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, 2018.

[28] G. Jia, Y. Zhu, G. Han, S. Chan, and L. Shu, "STC: an intelligent trash can system based on both NB-IoT and edge computing for smart cities," *Enterprise Information Systems*, vol. 14, no. 9-10, pp. 1422–1438, 2020.

[29] X. Lyu, C. Ren, W. Ni, H. Tian, and R. P. Liu, "Distributed optimization of collaborative regions in large-scale inhomogeneous fog computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 574–586, 2018.

[30] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," in *Proceedings of the First International Conference on E-Science and Grid Computing (E-Science'05)*, Melbourne, VIC, Australia, July 2005.

[31] J. Ren, G. Yu, Y. Cai, and Y. He, "Latency optimization for resource allocation in mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, no. 8, pp. 5506–5519, 2018.

[32] Z. Liu, X. Wang, D. Wang, Y. Lan, and J. Hou, "Mobility-aware task offloading and migration schemes in SCNs with mobile edge computing," in *Proceedings of the 2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6, Marrakesh, Morocco, April 2019.

[33] F. Wei, S. Chen, and W. Zou, "A greedy algorithm for task offloading in mobile edge computing system," *China Communications*, vol. 15, no. 11, pp. 149–157, 2018.

[34] Z. Xiao, X. Dai, H. Jiang et al., "Vehicular task offloading via heat-aware MEC cooperation using game-theoretic method," *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 2038–2052, 2020.