IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

SPECIAL SECTION ON MOBILE EDGE COMPUTING AND MOBILE CLOUD COMPUTING: ADDRESSING HETEROGENEITY AND ENERGY ISSUES OF COMPUTE AND NETWORK RESOURCES

Received June 21, 2019, accepted July 7, 2019, date of publication July 15, 2019, date of current version August 8, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2928582

A 'Joint-Me' Task Deployment Strategy for Load Balancing in Edge Computing

YUNMENG DONG[1], GAOCHAO XU[1], YAN DING[2], XIANGYU MENG[1], AND JIA ZHAO[2]

[1]College of Computer Science and Technology, Jilin University, Changchun 130012, China
[2]Jilin Province S&T Innovation Center for Physical Simulation and Security of Water Resources and Electric Power Engineering, Changchun Institute of Technology, Changchun 130012, China

Corresponding author: Jia Zhao (zhaiyj049@sina.com)

This work was supported in part by the National Key Research and Development Plan of China under Grant 2017YFA0604500, in part by the Jilin Provincial Industrial Innovation Special Foundation Project under Grant 2017C028-4, and in part by the Jilin Province Science and Technology Development Plan Project, Big Data Intelligent Vehicle Networking Service Platform, under Grant 20180623005TC.

ABSTRACT Task deployment has become a hot topic for load balancing in edge computing. In view of the problem that most of the hosts are overloaded in the edge computing, the central load is unbalanced. Much work focuses on the load balancing of the cloud data center or the short-term load balancing of edge data centers. In order to solve the host selection problem of task deployment in joint cloud data centers with edge computing while achieving the overall long-term load balancing, this paper utilizes that the deployment mode of joint cloud model, on this basis, proposes a deployment strategy HEELS based on the analysis of heuristic task clustering method and glowworm swarm optimization algorithm. Its main idea consists of two parts. First, the task with large resources in the current task set is filtered out by the clustering analysis, and the task offloading technology is exploited to upload the result to the cloud computing center for deployment and calculation. Then, the optimized GSO algorithm is exploited in the edge computing center, and the idea of SCA is combined into the optimization of step size so that the optimized GSO algorithm has an adaptive step size, achieving better global search ability in the early stage and better local convergence ability in the later stage. The experimental results show that compared with the existing research, HEELS realizes better load balancing effect and makes the joint datacenter more green and efficient.

INDEX TERMS Task deployment, load balancing, edge computing, joint cloud model, clustering analysis.

I. INTRODUCTION

Edge computing [1] is a promising and valuable research direction after distributed computing, grid computing and cloud computing. It is a hot topic in current research. Edge computing is a new architecture mode that extends computing, bandwidth, and other capabilities from the cloud to the edge of the network [2], where data processing and computing are carried out to reduce network operation and service delay. A large number of physical hosts are deployed in resource pools of edge data centers, and the remaining resources of physical hosts have been changing at any time. When the resource amounts of task requests submitted by users are greater than the remaining resource amounts of the currently deployed hosts, task deployment events will fail. When the resource amounts of task requests are close to the remaining resource amounts of the currently deployed hosts, the current tasks will be processed slowly. Not only that, but it also makes subsequent tasks unable to be deployed, and makes the load of the edge data center unbalanced, which cannot provide users with real-time calculation results while losing the advantage of edge computing.

At present, task deployment for load balancing has become a hot topic for joint cloud frameworks of combining edge computing centers with cloud computing centers [3]. In order to achieve joint load balancing between the edge computing center and the cloud computing center, it is necessary to carry out an efficient task deployment strategy. The selection of target host is the key to efficient task deployment. Deploying the user-requested tasks to the physical host with the best performance can improve the computational efficiency of edge computing and cloud computing, reduce the user's waiting time and system delay, and return the calculation results to the user in a short time, so as to achieve the long-term load

The associate editor coordinating the review of this manuscript and approving it for publication was Guanding Yu.

balancing of the joint cloud framework. However, current research on task deployment for load balancing is not perfect enough for the proposed joint cloud framework, and it cannot guarantee that tasks will be deployed to appropriate hosts every time.

In order to achieve efficient task deployment and load balancing in the joint cloud framework, this paper proposes a deployment approach based on analysis of heuristic task clustering and GSO optimization algorithm. First of all, the current task set is pre-processed, and the tasks requiring large amount of resources are filtered out by the clustering method. The task offloading technology is exploited to upload the result to the cloud computing center for deployment and calculation, and to a certain extent, it has the potential to achieve efficient task deployment and long-term load balancing of the joint cloud center. And then, the optimized GSO algorithm is exploited in the edge computing center, and the idea of SCA algorithm [4] is combined into the step size optimization, so that the GSO algorithm has an adaptive step size, achieves better global search ability in the early stage and better local convergence ability in the later stage.

This paper aims to achieve efficient task deployment and load balancing of joint cloud frameworks, to provide users with better computing capacity and achieve the best service performance of systems. The key to achieving the above goals is to enable the optimal deployment of the requested tasks so that the load between the edge computing center and the cloud computing center is more balanced, thereby improving the computational efficiency of the joint cloud framework. In this way, the high-quality service performance of systems can be provided to users. The proposed task deployment approach can not only efficiently find optimal physical hosts for deployed tasks, but also achieve joint load balancing between the edge computing center and the cloud computing center.

The main contributions of this study are as follows:

1) A novel joint architecture of ''cloud-edge'' is proposed to optimize deployment of tasks for load balancing from a global perspective combining edge computing and cloud computing.
2) The idea of SCA algorithm is integrated into the optimization of step size so that the optimized GSO algorithm has an adaptive step size, achieving precise deployment of tasks in edge computing centers.
3) A preprocessing method based on clustering analysis is designed to achieve optimal deployment of tasks and improve the execution efficiency of algorithm.

The rest of this paper is organized as follows. In the second part, the related work of task deployment approaches in current edge computing and cloud computing environment is briefly introduced. In the third part, we first briefly explain the premise of the problem raised in this paper, and then formalize the proposed problem. In the fourth part, the system architecture of task deployment in joint edge computing and cloud computing environment is designed firstly. Then the design and implementation process of the algorithm are introduced in detail. Finally, the source of the algorithm proposed in this paper is deeply analyzed. The fifth part gives the experimental results in detail, which prove that the proposed HEELS is effective and efficient. The sixth part summarizes the full text.

II. RELATED WORK

The purpose of task deployment is to achieve efficient computing performance of joint cloud frameworks by managing and deploying current computational tasks, and to make joint load balancing between edge computing centers and cloud computing centers. Compared to cloud computing, resources on edge servers are limited, dynamic, and heterogeneous. The problem proposed in this paper is to process tasks efficiently and improve the QoS (Quality of Service) of users by deploying the currently requested task set in the joint cloud framework. Finally, the joint load balancing between edge computing centers and cloud computing centers can be realized. Task deployment can be generally classified into three categories based on different goals: latency awareness, energy consumption awareness, and compute-intensive task offloading.

Latency awareness is an important performance metric for user experience. Souza VBC et al. in [5] proposed an architectural model in combination with cloud computing and fog computing, which minimizes service delay while ensuring that capacity requirements are met. In [6], Li CL et al. proposed a new task scheduling method under the edge computing environment, which combined the optimal placement of data blocks with the optimal task scheduling, not only reducing the computation delay and response time of tasks, but also improving the user experience of the edge computing center. However, this method only considers the problem of performance, and does not mention the energy consumption and load balance of edge computing centers. In [7], Islam T et al. made use of ford-fulkerson algorithm and priority-based queue to enable fog devices or networks based on edge data centers to rapidly process massive data or big data, thus achieving load balancing and efficient task deployment. In [8], Mach P et al. proposed a distributed cloud-aware power control algorithm for the application of delay-sensitive. In [9], Xu X et al. proposed a dynamic resource allocation method for load balancing in the fog computing environment - DRAM, which can effectively achieve efficient deployment of tasks and load balancing of fog computing nodes, and reduce service delay. In [10], Ebadifard F et al. proposed a static scheduling method based on particle swarm optimization algorithm, which assumes that the task has non-preemptive and independence, and utilizes load balancing technology to improve the performance of the basic particle swarm optimization method. Compared with the basic particle swarm optimization algorithm, the algorithm reduces the delay of task execution to a certain extent, improves resource utilization and performance, and maintains load balancing of the entire system.

Energy-aware task scheduling not only effectively improves users' QoS experience, but also is a kind of important methods to achieve load balancing in computing centers. In [11], Tang C et al. modeled the task scheduling problem at the end-user mobile device as an energy consumption optimization problem, while considering task dependency, data transmission and other constraint conditions such as task deadline and cost. In [12], Gao Y et al. proposed a multi-objective ant colony algorithm to solve the problem of the virtual machine placement, which aims to minimize total resource waste and power consumption while obtaining a set of non-dominated solutions (Pareto set). The experimental results showed that the algorithm can effectively place the virtual machine to the corresponding physical host while minimizing the total resource waste and power consumption. However, the algorithm only considers the cost of virtual machine deployment and does not mention the overall load balancing of the data center. In [13], Mishra S K et al. proposed a task-based virtual machine placement algorithm (ETVMC), with the goal of effectively assigning tasks to virtual machines and then assigning virtual machines to hosts, thus to minimize the allocated energy consumption, completion time and deployment task failure rate. In [14], Zhang K et al. designed a threshold-based scheduling scheme by considering energy consumption model and wireless channel model, and combined the multi-access characteristics of 5G heterogeneous network in [15], to design an EECO scheme, which obtained the minimum energy consumption under the delay constraint through joint optimization of offloading and wireless resource allocation. In [16], Sardellitti S et al. minimize the energy consumption of the overall user by jointly optimizing radio resources and computing resources.

The computing task offloading technology [17]–[19] solves the shortcomings of the device in terms of resource storage and computing performance, which not only reduces the pressure on the core network, but also reduces the delay caused by the transmission. In order to maximize the number of applications processed and meet the delay constraint, Zhao T et al. in [20] proposed a priority-based offload strategy, which defines several buffer thresholds for each priority level. If the buffer is full, the application will be transferred to the CC process, which utilizes recursive algorithms to find the optimal size of the buffer threshold. In [21], Deng S et al. considered the dependency relations among component services and aims to optimize execution time and energy consumption of executing mobile services, propose a novel offloading system to design robust offloading decisions for mobile services. In [22], Xu J et al. exploited deep learning to allocate resources and proposed a dynamic offloading scheme. In [23], Ketyko I et al. considered the load balancing among multiple nodes during computational offloading, and adopted knapsack model to carry out simulation verification for the proposed resource allocation scheme. In [24], Wang Y et al. developed partial computational offloading scheme to minimize the delay of application execution.

Based on the above research, this paper utilizes a joint deployment model. In the joint edge computing and cloud computing environment, the task is efficiently deployed by deploying the task set to the edge computing center and the cloud computing center under certain conditions. The task can be processed efficiently and the delay is minimal, thereby improving the user's QoS, and achieving joint load balancing between the edge computing center and the cloud computing center eventually.

III. THE PROPOSED PROBLEM AND ITS FORMALIZATION

A. PROBLEM STATEMENT

In edge computing centers, the system will deploy tasks to the hosts of edge computing center when users submit task requests. In general, the host will be randomly selected by the edge computing center for deployment. When the resource amount requested by the task is greater than the remaining resource amount of the physical host, the edge computing center cannot deploy the task. When the resource amount requested by the task is close to the remaining resource amount in the edge computing host, it will lead to the overload of the physical host, resulting in the decline of its service capacity and computing capacity, thereby causing the load of the edge computing center to be unbalanced. Obviously, in the face of large-scale task requests and limited computing capacity of the edge computing center, different deployment modes and deployment strategies will make the entire system have different load distribution. As a result, it has different execution efficiency and external computing service capability. Undoubtedly, optimal task deployment modes and strategies should enable load balancing between edge computing centers and cloud computing centers, as shown in Figure 1. Therefore, it is necessary to design and implement a high-efficiency, load-balanced task deployment model and strategy in the edge computing centers and the cloud computing centers.

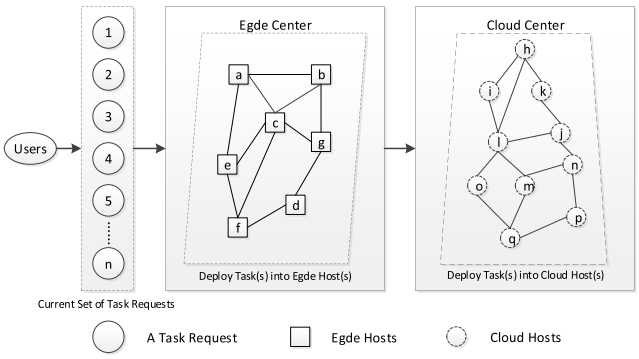


FIGURE 1. A joint deployment architecture for edge computing and cloud computing.

B. FORMALIZATION OF THE PROBLEM

There are n task requests accumulated in the time window Δt, which need to be deployed to an edge computing center composed of m physical hosts and a cloud

computing center composed of k physical hosts, respectively. There are i tasks to be deployed to the cloud computing center and n-i tasks to be deployed to the edge computing center. Therefore, this paper utilizes a two-dimensional solution vector $A = (a_1, a_2)$ to represent a task deployment solution, where a_1 represents the solution deployed to the edge computing center composed of m physical hosts, and a_2 represents the solution deployed to the cloud computing center composed of k physical hosts. Assume that in the same network environment, the edge computing center has m hosts available, and the cloud computing center has k hosts, which are heterogeneous, dynamic, and utilize space sharing allocation strategy. The problem can be described as follows. Finding the optimal physical hosts in the edge computing center and the cloud computing center for processing the current tasks. In this way, the edge computing center and the cloud computing center can rapidly deploy and calculate user task request set. By optimizing the task deployment in each Δt time, the problem of joint efficient task deployment and load balancing between the edge computing center and the cloud computing center is solved in the long run. A binary $P = \{TK_i, C\}$ is defined to describe the scenario of a cloud computing center. C represents a set of available hosts in a cloud computing center, $C(k) = \{c_1, c_2, \ldots, c_k\}$, and TK_i represents a set of tasks requested by users within a Δt time, $TK(i, \Delta t) = \{tk_1, tk_2, \ldots, tk_i\}$. At the same time, a four-tuple $Y = \{E, TK_{n-i}, L_c, L_{mem}\}$ is defined to describe the scenario of an edge computing center, E is represented as a set of available hosts, $E(m) = \{e_1, e_2, \ldots, e_m\}$, and the TK_{n-i} is represented as a set of user task requests within Δt time, $TK(n\text{-}i, \Delta t) = \{tk_1, tk_2, \ldots, tk_{n-i}\}$. L_c is the set of current CPU remaining of n hosts in set E at time t, $L_c(n-i) = \{\mathrm{L}_\mathrm{c}^1, \mathrm{L}_\mathrm{c}^2, \ldots, \mathrm{L}_\mathrm{c}^{n-i}\}$. L_{mem} is the memory remaining set of n hosts in set E at time t, $L_{mem}(n-i) = \{L_{mem}^1, L_{mem}^2, \ldots, L_{mem}^{n-i}\}$. The goal of this paper is to deploy tasks accumulated in Δt time to physical hosts in edge computing centers and cloud computing centers, so that tasks can be efficiently deployed and rapidly responded to users, thus achieving a long-term load balancing between edge computing centers and cloud computing centers to a certain extent. Therefore, the optimization goal can be formalized into the following formula (1):

$$V = \sqrt{\frac{1}{m} * \sum_{i=1}^{m} \left(u_e^i - \overline{u_e}\right)^2} + \sqrt{\frac{1}{k} * \sum_{j=1}^{k} \left(u_c^j - \overline{u_c}\right)^2} \quad (1)$$

In order to achieve efficient deployment and load balancing of tasks in the joint cloud framework, the task set is preprocessed before deployment. Screen out the tasks with larger resource requirements by the clustering method to ensure that the remining tasks can be efficiently processed by the edge computing center. And then these tasks gotten by the clustering method are deployed to the cloud computing center and thus to use powerful computing abilities of the cloud computing center to process them. In this paper, the idea of clustering analysis is exploited to compare the similarity between tasks with a given threshold. The tasks with the above attributes are composed of a new set and uploaded to the cloud computing center for deployment. The similarity function is defined as follows:

$$d(tk_i, tk_j) = \sqrt{\sum_{k=1}^{d} \left(a_i^k - a_j^k\right)^2} \quad (2)$$

$$SD\left(tk_i, tk_j\right) = \frac{1}{d\left(tk_i, tk_j\right)} \quad (3)$$

where a_i^k and a_j^k are the k-th attribute of task i and task j respectively, and $SD(tk_i, tk_j)$ is the similarity between task i and task j.

After preprocessing, both the resource amount and time delay of the tasks in the set can be satisfied and processed by the host in the edge computing center. In this case, the remaining task sets need to be deployed to the edge host. In this paper, the idea of GSO algorithm is exploited to deploy tasks in a set efficiently and reasonably. However, GSO algorithm utilizes fixed step size and has its own defects in the iteration process. There are some problems such as low accuracy, local optimum and slow convergence speed, etc. In order to solve these problems, this paper proposes a strategy of adaptive step size based on sine and cosine. To a certain extent, this strategy can make HEELS approach avoid falling into local optimum too early, and the step size is adjusted adaptively with the increase of iteration times, so that the algorithm can obtain a more accurate solution at the later stage. The sine and cosine functions are defined as follows:

$$s_i(t+1) = \begin{cases} s_i(t) + r_1 \times \sin(r_2) \times \left| r_3 X_{gbest(t)} - X_{i,best(t)} \right| \\ \mathrm{r}_4 < 0.5 \\ s_i(t) + r_1 \times \cos(r_2) \times \left| r_3 X_{gbest(t)} - X_{i,best(t)} \right| \\ \mathrm{r}_4 \geq 0.5 \end{cases} \quad (4)$$

$$\mathrm{r}_1 = a\left(1 - \frac{t}{T}\right) \quad (5)$$

where t is the current number of iterations, $s_i(t)$ is the step size of the i-th individual of the t-th iteration, $X_{gbest(t)}$ is the global optimal position so far, and $X_{i,best(t)}$ is the distance of the i-th individual to the current optimal position. There are four main parameters in formula (4), where r_1 is the amplitude adjustment coefficient of sine cosine, as defined by equation (5), a is the normal number, T is the maximum number of iterations, and r_1 determines the movement direction of the next iteration of step size. $r_2 \in [0,2\pi]$, $r_3 \in [0,2]$, and $r_4 \in [0,1]$ are random numbers. r_2 determines the moving distance by the next iteration of the step size. r_3 is the global optimal individual weight coefficient. r_4 is the discrimination coefficient. When r_4 is less than 0.5, sine function is used for iterative optimization, on the contrary, the cosine function is used for optimization and update.

In order to find hosts with the best performance, this paper reflects the fitness through the residual load rate of

the resources in each host node. The current remaining resource L_i (CPU and memory usage status) of each node is defined as follows:

$$L_i = \alpha L_c + \beta L_{mem} \tag{6}$$

$$\alpha + \beta = 1 \tag{7}$$

where L_c represents the amount of remaining CPU resource of the host, L_{mem} represents the amount of remaining memory of the host, α is the weight of CPU and β is the weight of memory. α and β are determined and obtained by BP neural network learning. The state of residual resource of a single node can be calculated by formula (6) and (7), and the total residual resource of available nodes in the edge computing center can be denoted as follows:

$$Q_i = \sum_{i=1}^{n} L_i \tag{8}$$

where Q_i is the total residual load of all the computing nodes in the edge computing center, and the residual load rate G_i is the ratio of the residual load of the computation node i to the total residual load, that is:

$$G_i = \frac{L_i}{Q_i} \tag{9}$$

In order to deploy the task set obtained by clustering to the cloud computing center, it is necessary to ensure that the available resources of physical hosts of the cloud computing center can accommodate the current task set. The remaining amount W of the physical host resources of the cloud computing center can be calculated by formula (8). The total amount of resources of the current task set is defined as follows:

$$L_{Treq} = \sum_{i=1}^{e} \left(\alpha R_c^i + \beta R_{mem}^i \right) \tag{10}$$

where L_{Treq} is the amount of total resource of tasks in the set, R_c^i is the CPU resource required by task i, and R_{mem}^i is the memory resource required by task i.

As mentioned above, the proposed approach is exploited to find the best host for the tasks in the set to process and thereby achieve the joint load balancing between the edge computing center and the cloud computing center. The detailed algorithm process will be given below.

IV. A HEURISTIC APPROACH FOR EFFICIENT TASK DEPLOYMENT

A. SYSTEM ARCHITECTURE DESIGN

Figure 2 depicts the system architecture of the proposed joint cloud framework consisting of the edge computing center and the cloud computing center. It shows the interaction between the proposed HEELS approach and other entities, and shows the important role of HEELS in the whole architecture. Firstly, the system preprocesses tasks through the preprocessor to get a new task set and deploy it to the cloud computing center. After the preprocessing is completed, information such as the current task set and the hosts

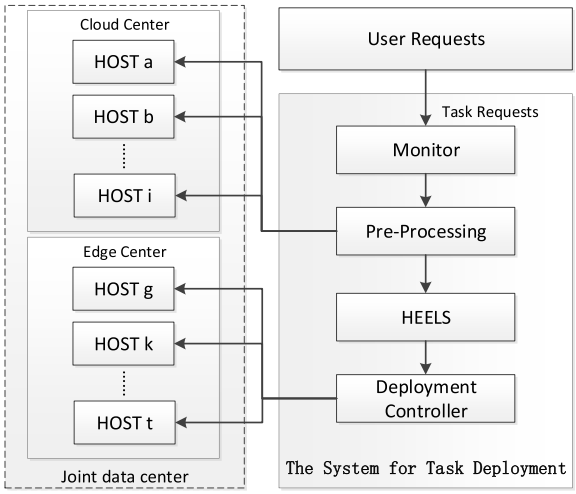


FIGURE 2. The view of HEELS's architecture.

in the edge computing center are obtained from the monitor. The information obtained is transmitted through the monitor, and the deployment strategy is generated by using the HEELS approach. Finally, the deployment strategy is applied to the deployment controller, and the task set received in the Δt time is deployed to the corresponding edge hosts through the HEELS deployment strategy.

B. MAIN IDEA OF HEELS

The proposed HEELS approach in this paper is a heuristic task deployment strategy based on clustering analysis and optimization of GSO algorithm for long-term load balancing of joint cloud framework combining the edge computing center with the cloud computing center, which is exploited to deploy collected tasks to hosts of the edge computing center and the cloud computing center, as shown in Figure 3. First of all, the idea of clustering is exploited to actualize task preprocessing. The n tasks in the task set are regarded as n objects to be clustered, and the task set requiring a large amount of resources are screened out by means of clustering and processed with the powerful computing power of cloud computing centers. Secondly, by introducing the mathematical model of the SCA algorithm into the optimization of the step size of GSO algorithm, the improved GSO algorithm has an adaptive step size to achieve better global search ability at the early stage and better local convergence ability at the later stage. Finally, when the maximum number of iterations is reached, the global optimal solution is obtained. Implementation details of HEELS are given below.

C. IMPLEMENTATION OF HEELS

This section will detail the implementation of the proposed HEELS algorithm.

Step1: monitor users' task requests and initialize related parameters of HEELS. In the initialization stage, the system collects the task set in Δt time, and the task set is taken as the

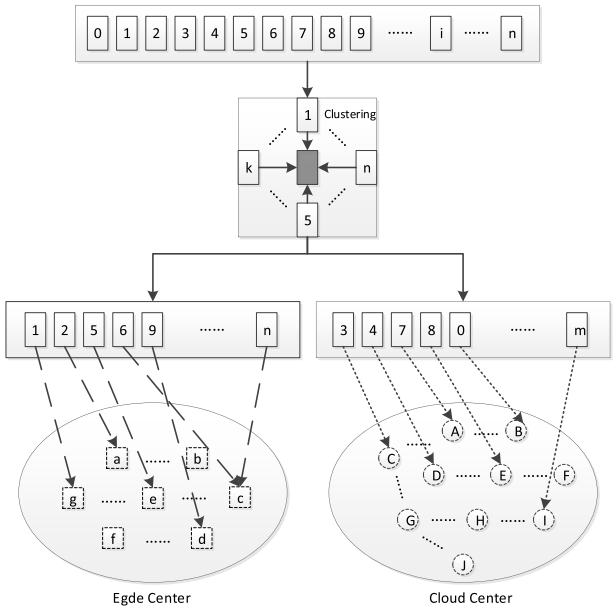


FIGURE 3. The process of HEELS task deployment.

object of the current processing problem. The core algorithm of HEELS will be executed once in each Δt time, and the set of tasks collected by the system during execution will be the next processing problem. As mentioned earlier, the number of tasks in the set, the number of hosts in the edge computing center and the cloud computing center in one Δt time are represented as n, m, k, respectively.

Step2: preprocessing of the initial set of tasks. After the relevant parameters are initialized, the current set of tasks needs to be preprocessed to ensure optimal deployment of the current tasks. Define an empty set $Q = \{\}$. It is known that there are n tasks $TK = \{tk_1, tk_2, \ldots, tk_n\}$ in the set. The requested resource amount L_t of a task is regarded as the attribute of the task, and the resource amount of the task in the set is arranged in descending order so that tk_j represents the task with the largest resource amount. Therefore, tk_j is selected as the cluster center in this paper. Then the similarity between any other task tk_i in the set and the cluster center tk_j can be calculated by formula (2) (3) as follows:

$$SD\left(tk_i, tk_j\right) = \frac{1}{\sqrt{\left(L_{ti} - L_{tj}\right)^2}} \tag{11}$$

where L_{ti} is the resource amount of task i and L_{tj} is the resource amount of task j, $L_{ti} \neq L_{tj}$, $SD(tk_i, tk_j)$ is the similarity between tasks i and j. The similarity between tk_j and other tasks in the set is calculated by using tk_j as the clustering center, and the threshold U is given according to the similarity value. If the similarity value SD is larger than the threshold U, this task is added to the empty set Q, which is the final clustering result, ie $Q = \{tk'_1, tk'_2, \ldots, tk'_e\}$, where $e \leq n$. The total remaining resource amount W of the cloud computing center and the total resource requirement amount L_{Treq} of the set Q can be calculated by the formula (8) and (10). If W is less than L_{Treq}, then the cloud computing center cannot accommodate the Q set, resulting in failures of task deployment events. At this point, the threshold U needs to be adjusted to form a new Q set, and the above process is repeated until W is greater than L_{Treq}. When W is greater than L_{Treq}, upload the Q set to the cloud computing center for deployment. The remaining tasks in the TK form a new task set H. If the task set Q is larger than the set H, the number of tasks in the set Q is large and the amount of resources is large. Due to limited computing capacity of edge computing center, deployment failure and load unbalancing of edge nodes may be caused. In this case, the tasks in the set Q are deployed to the cloud computing center for processing through the task offloading. If the task set Q is less than the set H, the set Q has the characteristics of small number and large amount of resources. If it is deployed in the edge center, the computational burden of edge nodes will be increased. In this case, the task offloading technology is used to deploy tasks in the set Q to the cloud center, which improves the computing capacity of the edge center and reduces the delay of the computation tasks. At the end of the preprocessing process, task set H is used as the initial population in the HEELS algorithm.

Step 3: The core iteration process of HEELS algorithm.

(1) Initialization. Initialize the maximum iteration number *Itermax*, the initial population of glowworm, the initial search space of each glowworm and the luciferin value carried by itself, and other relevant parameters.

(2) Calculate the luciferin value of the individual. Update the luciferin value of each glowworm individual f_i $(i = 1, 2 \ldots N)$ according to formula (12):

$$f_i(t) = (1 - \rho) f_i(t - 1) + \gamma J(x_i(t)) \tag{12}$$

where $f_i(t)$ represents the luciferin value of glowworm i at the *t-th* iteration, i.e. the brightness of glowworm i; ρ represents the brightness decay constant ($0< \rho <1$), and $(1\text{-}\rho)$ represents the brightness decay rate, which is used to control the proportion of past experience so that glowworm forgets the past non-optimal solution in iteration; γ represents the proportional constant, which is used to control the empirical proportion of the search solution in the iteration. $J(x_i(t))$ is the fitness value. As mentioned above, the residual load rate of the host is used as the fitness function in this paper, which can be obtained according to formula (6), (7), (8), (9).

$$J(x_i(t)) = \frac{\alpha L_c^i + \beta L_{mem}^i}{\sum_{i=1}^{n} \alpha L_c^i + \beta L_{mem}^i} \tag{13}$$

The luciferin value of the glowworm individual can be calculated according to formula (12), (13).

$$f_i(t) = (1 - \rho) f_i(t - 1) + \gamma \frac{\alpha L_c^i + \beta L_{mem}^i}{\sum_{i=1}^{n} \alpha L_c^i + \beta L_{mem}^i} \tag{14}$$

(3) Look for neighborhood $N_i(t)$. Each glowworm chooses individuals whose luciferin value is higher than itself in its

dynamic decision domain radius $r_d^i(t+1)$ to form its domain set $N_i(t)$, where $0 < r_d^i(t+1) < r_s$. r_s is the perceptual radius of the individual glowworm. The neighborhood set $N_i(t)$ is:

$$N_i(t) = \{j : ||x_j(t) - x_i(t)|| < r_d^i(t); f_i(t) < f_j(t)\} \quad (15)$$

(4) Determine the movement direction. The movement direction of glowworm i is affected by the glowworm with the larger luciferin value in its neighborhood. The larger the luciferin value is, the more attractive it will be. Correspondingly, the probability which glowworm will choose to move toward it with is larger. Suppose glowworm i moves towards glowworm j with the highest probability, then $J = \max(p_i)$ where $p_i = (p_{i1}, p_{i2}, \ldots, p_{iN_i(t)})$, the selection probability of glowworm i moving to j is:

$$p_{ij}(t) = \frac{f_j(t) - f_i(t)}{\sum_{k \in N_i(t)} f_k(t) - f_i(t)} \quad (16)$$

Among them, the denominator is the brightness of all glowworms in the neighbor set of glowworm i, and the numerator is the brightness of neighbor j. The higher $P_{ij}(t)$ value of a neighbor in the neighbor of glowworm i, the greater the probability of selecting it as the target of glowworm i. In this paper, roulette is used to select the direction of individual movement, and then one will move and update the position.

(5) Update the glowworm position. The moving step is adaptively adjusted according to formula (4), (5), and then the step is brought into the formula (17) to update the position of the glowworm. Therefore, the position update of the adaptive step size of glowworm individuals is shown in formula (18).

$$X_i(t+1) = X_i(t) + s\left(\frac{X_j(t) - X_i(t)}{\| X_j(t) - X_i(t) \|}\right) \quad (17)$$

$$X_i(t+1) = \begin{cases} X_i(t) + \left(\begin{array}{l} s_i(t) + r_1 \times \sin(r_2) \times \\ |r_3 X_{gbest(t)} - X_{i,best(t)}| \end{array}\right) \\ \times \left(\dfrac{X_j(t) - X_i(t)}{||X_j(t) - X_i(t)||}\right) \mathrm{r}_4 < 0.5 \\ X_i(t) + \left(\begin{array}{l} s_i(t) + r_1 \times \cos(r_2) \times \\ |r_3 X_{gbest(t)} - X_{i,best(t)}| \end{array}\right) \\ \times \left(\dfrac{X_j(t) - X_i(t)}{||X_j(t) - X_i(t)||}\right) \mathrm{r}_4 \geq 0.5 \end{cases} \quad (18)$$

(6) Update the value of dynamic decision domain radius of the glowworm. Update the radius of the decision domain according to formula (19).

$$r_d^i(t+1) = \min\left\{r_s, \max\left\{0, r_d^i(t) + \mu\left(n_t - |N_i(t)|\right)\right\}\right\} \quad (19)$$

where μ is the update rate of the decision domain and n_t is the threshold of the number of neighbor individuals.

(7) Determine whether the number of iterations exceeds the maximum value *Itermax*. If not, go to step (2); otherwise, go to step (8).

(8) Output the global optimal solution.

Step 4: Repeat the above process.

The pseudo-code of HEELS's algorithm is as follows. From the perspective of the overall framework, this paper proposes a novel architecture of joint deployment based on the existing deployment mode, which effectively integrates cloud computing and edge computing, that is, "cloud-edge" as a whole. On this basis, an effective deployment strategy has been utilized to achieve efficient computing power and joint load balancing effect of the proposed joint architecture to some certain extent. It has abilities in effectively mitigating the computational performance degradation and load imbalance of cloud computing centers and edge computing centers caused by the irregular deployment of tasks. The shortcomings of existing deployment models are overcome from a long-term perspective. As a result, the proposed joint deployment framework is effective and efficient for a long time, and the cooperative processing and the optimization of "Cloud-Edge" are realized.

Algorithm 1 Algorithm HEELS

Input: Current set of tasks, Current set of edge servers, Current set of cloud servers, Maximum Iterations *Itermax*, current L_c, current L_{mem};
Output: final deployment solution vector V;
1: $Q = \{\}$, $TK = \{tk_1, tk_2, \ldots, tk_n\}$, $E = \{e_1, e_2, \ldots, e_m\}$, $C = \{c_1, c_2, \ldots, c_k\}$,V=*NULL*;
2: tk_j is the task with the Maximum resources in the set and selected as the task clustering center;
3: $Q = Q \cup tk_j$;
4: for each $tk_i \in TK$ do
5: $SD(tk_i, tk_j)$ is obtained according to the formula (11);
6: if $SD(tk_i, tk_j) > U$ then
7: $Q = Q \cup tk_i$;
8: end if
9: end for
10: Deploy Q to the set C in cloud computing center
11: The remaining tasks in TK form a new set H
12: set $y = 1$
13: while current iteration number $y <$ *Itermax* do
14: for each $e_i \in E$ do
15: $G_i = \dfrac{\alpha L_c^i + \beta L_{mem}^i}{\sum_{i=1}^{n} \alpha L_c^i + \beta L_{mem}^i}$ // Fitness function
16: Calculate the luciferin value $f_i(t)$ of individual i
17: for each $n \in N_i(t)$ do
18: The individual's moving probability P_{ij} is obtained according to the formula (16);
19: end for
20: According to formula (4) and (5), the moving step size s of each individual is calculated
21: Update X_i // The latest location of glowworm i
22: Update r_d // The dynamic decision domain radius
23: end for
24: $y = y+1$
25: end while
26: return V;

From the perspective of adaptability of HEELS algorithm, this paper utilizes the SCA algorithm with strong mathematical foundation to determine the optimal step size of each iteration. The reason why the SCA algorithm is used in this paper to optimize the step size of GSO process is mainly that the traditional swarm intelligence glowworm algorithm uses fixed step size in solving global optimization problems, which has the problems of low accuracy, local optimum and slow convergence speed. By introducing the idea of SCA algorithm, the original fixed step size is optimized to an adaptive variable step size, which can make the initial stage of the algorithm have a larger step size and expand the search space of HEELS. As the number of iterations increases, the step size rapidly converges to the optimal state by updating parameters such as sines, cosines functions and amplitudes. It can be seen through analyzing the proposed algorithm that the computational complexity of the algorithm is $O(n^2)$.

From a micro point of view, this paper utilizes the idea of clustering in the initial stage to realize a preprocessing, so that the algorithm makes full use of the advantages of cloud computing and edge computing. A task is regarded as a clustering object, and the task set requiring large amount of resources is screened out through clustering. The powerful computing power of cloud computing center is utilized to deal with the current tasks, which avoids the deployment failure caused by the inability of edge computing centers and the load imbalance of edge computing centers to some extent. The task set requiring small amount of resources is deployed in edge computing centers to make full use of the real-time advantages of edge computing. In edge computing centers combining clustering with bionic swarm intelligent glowworm algorithm, the remaining task set after clustering exactly right simulates the initial population of glowworm, and then the glowworm algorithm is used to match the tasks in the set to the hosts synchronously in the current edge computing center, and the seamless connection between the clustering method and the glowworm algorithm is achieved, and thus the scientific problem raised by this paper is addressed.

V. PERFORMANCE EVALUATION AND ANALYSIS

In this part, we mainly evaluate the performance of the proposed HEELS algorithm to verify and test its effectiveness and efficiency. Firstly, the setting of algorithm simulation environment is introduced. Then, a metric to verify the effectiveness of HEELS is given. Finally, the OTS deployment strategy [6], First Fit Decreasing (FFD) algorithm and the proposed HEELS deployment strategy are simulated and compared through the maximum completion time (MakeSpan), the success rate of deployment tasks, throughput and load balancing degree.

The experimental results show that compared with FFD and OTS, HEELS enables the edge computing center to improve the success rate of deployment tasks, shorten the maximum completion time (MakeSpan) of tasks, greatly improve the system throughput and provides better service performance for users. On this basis, the proposed HEELS algorithm makes the edge computing center have a better load balancing effect.

This paper uses Python language programming to realize the simulation experiment on HEELS. In Python environment, the edge computing center is simulated and different number of task requests are initiated to the computing resource pool, and the amounts of resources required for these requests are different, that is, the CPU and memory of the physical host are different. The proposed HEELS approach periodically calls and obtains the resource information and status of physical hosts in the edge resource pool.

A. EXPERIMENTAL METRICS

The experimental metrics of the HEELS deployment strategy proposed in this paper mainly include the success rate of deployment tasks, the maximum completion time (MakeSpan) of tasks, throughput and load balancing degree.

The maximum completion time (MakeSpan) is the total execution time of all task requests processed by the system in Δt time, as shown below:

$$MakeSpan = \max_{i \in N} \{CT_i\} \tag{20}$$

CT_i is the completion time of the *i-th* task.

Deployment task success rate. If task $i's$ completion time is less than its own deadline, it will be regarded as a successful deployment within the deadline constraint; otherwise, the deployment fails. Then the calculation formula of the task deployment success rate φ is as follows:

$$\varphi = \frac{n_{success}}{N} \times 100\% \tag{21}$$

$n_{success}$ is the number of successful deployment and N is the total number of tasks.

In this paper, throughput is used to evaluate the service performance of the system. By measuring the total number of tasks that the whole system can complete in a given time, the more tasks processed per unit time, the better the service performance of the system is proved. Throughput calculation formula is as follows:

$$F = V_u \times R/T \tag{22}$$

where V_u is the number of users, R is the number of task requests submitted by each user, and T is time.

Load balancing degree. Load balancing degree refers to the balance degree of the load distributed on each processing nodes of the parallel system. Load balancing degree is an important factor affecting the parallel efficiency. When the whole system has more tasks, the load on each node may produce unbalanced phenomenon, which will reduce the utilization rate of the whole system. The CPU utilization variance of each physical host is used to represent the load balancing degree, u_i is used to represent the current CPU utilization of physical host resource i, and m is the number of physical hosts. The average CPU utilization rate at any time is calculated by formula (23). Then load balancing degree is

obtained as formula (24):

$$\overline{u} = \frac{\sum_{i=1}^{m} u_i}{m} \tag{23}$$

$$B = \sqrt{\frac{1}{m} * \sum_{i=1}^{m} (u_i - \overline{u})^2} \tag{24}$$

The above value is used to measure the load balancing degree of the edge computing center, and the goal is to make the value of B as small as possible.

Through the above four experimental metrics to verify the efficiency of the algorithm, the edge center can achieve efficient external service performance, and reach long-term load balancing effect of the edge center while improving computational efficiency.

B. COMPARISON IN MAKESPAN

In this set of experimental scenarios, the maximum completion time (MakeSpan) of the proposed HEELS approach is compared with FFD and OTS. The MakeSpan of the current deployment task can be obtained by formula (20). The MakeSpan of the three deployment strategies can be compared by increasing the number of tasks. As shown in Figure 4, as the number of requested tasks increases, the MakeSpan values of the three deployment strategies will increase. When the number of tasks is 20, the three deployment strategies have the same MakeSpan. The main reason is that the three deployment strategies have a little number of tasks in the initial stage, and time complexity of processing tasks is approximately the same, so they have the same MakeSpan. The MakeSpan increment of HEELS is less than those of FFD and OTS when the number of tasks is in the [40], [80] range. When the number of tasks is 100, the proposed HEELS has the smallest MakeSpan compared to FFD and OTS. The main reason is that the FFD deployment strategy is to arrange the requested tasks in descending order of the task size and deploy them to the hosts in the edge computing center in order, which has better processing performance at the beginning. With the increase of the number of requested tasks, its processing ability gradually weakens, so the task completion time will inevitably increase. The OTS deployment strategy allocates the current task set by calculating the similarity value between the task and the host. With the increase of the number of requested tasks, the computational complexity between the physical host and the task is bound to increase, the processing capability of the task is gradually weakened, and the required time will increase, but it is always less than FFD. Since HEELS optimizes the step size of the individual by using the SCA algorithm in the iteration process, it can deploy the requested tasks to the optimal hosts each time. With the increase of task requests, the processing time of HEELS will also increase, but less than those of FFD and OTS under the same task requests.

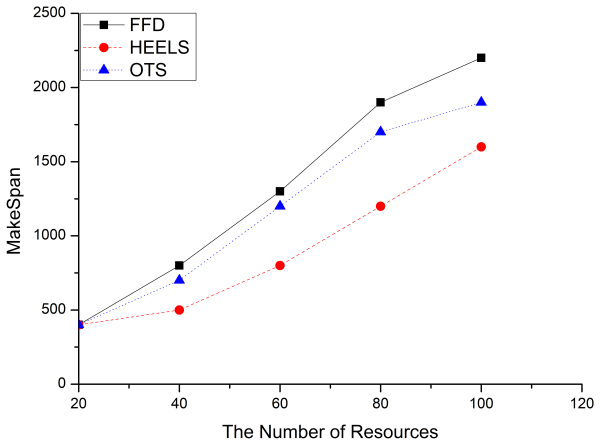


FIGURE 4. Comparison of FFD, OTS, and HEELS on MakeSpan.

TABLE 1. Success rate of task deployment.

Success rate of task deployment					
	Number of tasks (0-100)				
	20	40	60	80	100
FFD	1	1	1	1	1
HEELS	1	1	1	1	1
OTS	1	1	1	1	1
	Number of tasks (100-1000)				
	200	400	600	800	1000
FFD	1	1	0.98	0.818	0.75
HEELS	1	1	1	0.95	0.9
OTS	1	1	1	0.85	0.8
	Number of tasks (1000-10000)				
	2000	4000	6000	8000	10000
FFD	0.556	0.472	0.35	0.3	0.101
HEELS	0.82	0.652	0.52	0.478	0.25
OTS	0.7	0.531	0.35	0.32	0.15

C. COMPARISON IN DEPLOYMENT SUCCESS RATE

In this set of experimental scenarios, HEELS is compared with FFD and OTS for the success rate of deployment tasks. The number of different tasks under three orders of magnitude is selected, and the success rate can be calculated by formula (21). Table 1 shows the different success rates generated by the three strategies. When the number of tasks is in the [0-100], they have the same success rate. With the increase of the number and magnitudes of task, the deployment success rates of the three strategies gradually decrease. When the number of tasks is in the [100-1000], the number of tasks is 600, the success rate of FFD begins to decline, while HEELS and OTS remain unchanged. When the number of tasks is greater than 800, the success rate of HEELS and OTS is slowly decreasing, and HEELS always higher than those of FFD and OTS. When the number of tasks is in the [1000-10000], the success rate of deployment tasks of the three strategies is gradually reduced, and the HEELS has a higher success rate, OTS is second, and the FFD is the

worst. This is because HEELS preprocesses the task set in the early stage. Based on the attributes of tasks, one can filter out the tasks with large amount and upload them to the cloud computing center for deployment. while the rest of the tasks according to the optimization of the improved GSO algorithm with global search optimization are deployed to end hosts. After the above-mentioned processing, the efficient deployment of tasks is guaranteed to a certain extent, and the success rate of deployment tasks is improved. FFD sorts tasks according to the size of current task resources, and then deploys current tasks separately. Batch task requests and the time complexity of FFD are greatly increased, which will reduce the success rate of task deployment to a certain extent. The OTS only deploys tasks to the host based on the similarity between the task and the host. The information about the remaining resources of the physical hosts in the resource pool cannot be obtained in real time, and the current tasks cannot be deployed timely. The experimental results show that HEELS can effectively improve the success rate of task deployment.

D. COMPARISON IN THROUGHPUT

In this set of experimental scenarios, HEELS is compared with FFD and OTS for throughput. The system throughput is taken as a measure of the service performance of the system. The current throughput of the system can be calculated by formula (22). As shown in Figure 5, FFD enables the system to have better throughput in the initial stage, and the throughput of the system grows slowly as time goes by. HEELS and OTS make the system throughput increase continuously. OTS has a greater initial throughput than HEELS. As time increases, the growths of the two strategies tend to be stable. However, the throughput of HEELS is always greater than that of OTS since the task preprocessing and heuristic optimization strategy of HEELS in the initial stage enable tasks to be deployed and responded rapidly, so that the system has the better throughput. FFD deploys the current task set rapidly according to the packing problem, so it has high throughput at the beginning. But with the increase of time and number of tasks, its adaptive ability is limited, and the throughput increment is always smaller than the other two strategies. OTS determines task deployment policies based on the similarity between the host and the task, which has a potential for long-term efficient deployment. By comparing the service performance of the edge center system, it can be concluded that the deployment strategy of HEELS has better stability and efficiency.

E. COMPARISON IN LOAD BALANCING DEGREE

In this experimental scenario, HEELS is compared with FFD and OTS for load balancing degree. The load balancing degree value of the current edge center is calculated by the formulas (23) and (24). The smaller the value is, the more balanced the system load is. As can be seen from Figure 6, OTS has a better initial load balancing degree, followed by FFD, and the HEELS load balancing effect is the worst. When $t = 300$, the HEELS load balancing degree is less than

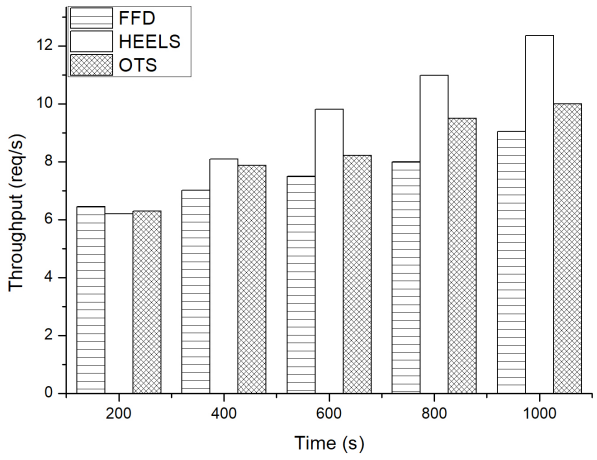


FIGURE 5. Comparison of FFD, OTS, and HEELS on throughput.

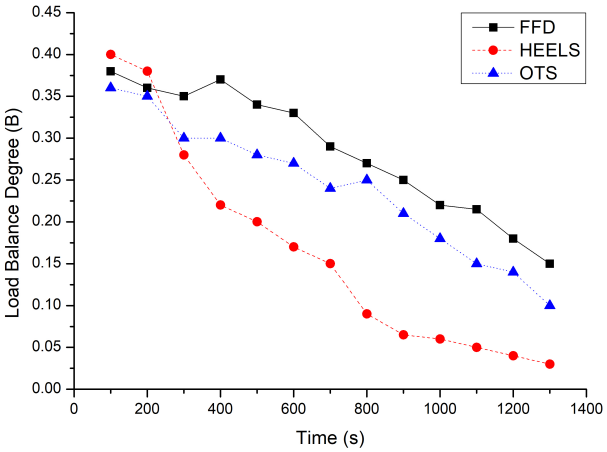


FIGURE 6. Comparison of FFD, OTS, and HEELS on load balancing degree.

OTS and FFD. When $t > 300$, the load balancing degree of the three task deployment strategies is gradually decreasing, and the load balancing degree of HEELS is always less than FFD and OTS in the process of descending. OTS is somewhere in between, and FFD performance worst. This is because OTS calculates the cosine between each task to be processed and hosts, then deploys the task according to the cosine value. When the number of tasks is too large, the real-time load information of the host cannot be guaranteed to be obtained. FFD is random and cannot guarantee that tasks will be deployed to the optimal host every time. The proposed HEELS approach can pre-process tasks in the initial stage of the algorithm, which can deploy the tasks requiring large amount of resources and high performance requirements to the cloud center, and then through heuristic optimization, it ensures that tasks are deployed to the hosts with the best performance to a certain extent, so each node has achieved good load balancing effect. The experimental results show that HEELS has better load balancing effect, so that the load balancing optimization of the edge computing center can be realized more effectively.

VI. CONCLUSION AND FUTURE WORK

Based on the joint deployment of edge computing and cloud computing, this paper proposes a novel heuristic task deployment approach HEELS based on clustering analysis and GSO optimization algorithm, and gives its main idea, specific implementation and performance evaluation. First of all, the task analysis process adopts the heuristic idea based on clustering, which filters out the tasks with large amount of requested resources by clustering method, and uploads the results to cloud computing center for deployment and calculation by using task offloading technology. To a certain extent, it has the potential to achieve efficient task deployment and long-term load balancing of joint edge computing centers and cloud computing centers. Then, an improved GSO algorithm is presented to deploy the tasks in the edge center, and the idea of SCA algorithm is integrated into the GSO algorithm. The mathematical model based on sine and cosine function of SCA algorithm is used to fluctuate outwards or inwards towards the optimal solution, which can explore different regions of the step size space, so that the GSO algorithm has an adaptive step size. By using the SCA algorithm to calculate the adaptive step size of GSO individuals, the GSO algorithm can get the global optimal solution vector faster. It makes the GSO method have better global search ability at the early stage and better local convergence ability at the later stage. It also shows that the improved GSO approach has a strong mathematical basis.

In order to evaluate the proposed HEELS approach, several simulation experiments are carried out in this paper. Firstly, in the comparison experiment of HEELS, OTS and FFD on the total task completion time, HEELS has a smaller maximum completion time under the same conditions. Secondly, HEELS has better load balancing effect than FFD and OTS in the comparison experiment of load balancing degree. Thirdly, in the comparison experiment of the system service performance demonstrated by the edge computing center and the cloud computing center after deploying the tasks through the three deployment strategies, it is proved that the proposed HEELS approach has better system service performance than the other two. Finally, the deployment success rates of FFD, OTS and HEELS are compared. The experimental results show that compared with FFD and OTS, HEELS improves the success rate of deployment tasks, and it is more efficient and reasonable to deploy tasks. Through four sets of comparison experiments, the proposed HEELS approach is more efficient in processing real-time task requests and enabling long-term load balancing between the edge computing center and the cloud computing center.

In HEELS, there are some open problems that need to be further studied and the empirical problems which require a large number of experiments to gradually obtain a better solution. Among them, the value α of CPU weight and the value β of memory weight are empirical problems, and multiple experiments are needed to obtain the optimal values such that $\alpha + \beta = 1$. In this paper, all parameters are set to the appropriate values.

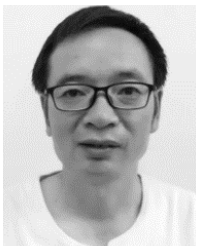
In order to further improve the performance of joint deployment of edge computing and cloud computing, it is planned to carry out research from multi-dimensional heuristic task analysis in the next step, and deploy the offloaded tasks to the cloud computing platform with certain strategies on the premise of reducing complexity. The joint deployment model has ability to simultaneously ensure that tasks are deployed to edge computing centers and cloud computing centers according to different policies, thus maximizing overall benefits for the joint cloud framework. In the future work and experiments, the empirical questions and open questions proposed in this paper will be further studied. If possible, we will study and implement the proposed task analysis and deployment approach in a real joint edge computing and cloud computing environment, evaluate its performance and efficiency, and verify the joint load balancing effect.

REFERENCES

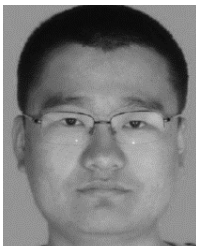
[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[2] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.
[3] W. Chen, B. Liu, H. Huang, S. Guo, and Z. Zheng, "When UAV swarm meets edge-cloud computing: The QoS perspective," *IEEE Netw.*, vol. 33, no. 2, pp. 36–43, Mar./Apr. 2019.
[4] S. Mirjalili, "SCA: A sine cosine algorithm for solving optimization problems," *Knowl.-Based Syst.*, vol. 96, pp. 120–133, Mar. 2016.
[5] V. B. C. Souza, W. Ramírez, X. Masip-Bruin, E. Marín-Tordera, G. Ren, and G. Tashakor, "Handling service allocation in combined fog-cloud scenarios," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–5.
[6] C. Li, J. Bai, and J. Tang, "Joint optimization of data placement and scheduling for improving user experience in edge computing," *J. Parallel Distrib. Comput.*, vol. 125, pp. 93–105, Mar. 2019.
[7] T. Islam and M. M. A. Hashem, "Task scheduling for big data management in fog infrastructure," in *Proc. IEEE 21st. Int. Conf. Comput. Inf. Technol. (ICCIT)*, Dec. 2018, pp. 1–6.
[8] P. Mach and Z. Becvar, "Cloud-aware power control for real-time application offloading in mobile edge computing," *Trans. Emerg. Telecommun. Technol.*, vol. 27, no. 5, pp. 648–661, May 2016.
[9] X. Xu, S. Fu, Q. Cai, W. Tian, W. Liu, W. Dou, X. Sun, and A. X. Liu, "Dynamic resource allocation for load balancing in fog environment," *Wireless Commun. Mobile Comput.*, vol. 2018, Apr. 2018, Art. no. 6421607.
[10] F. Ebadifard and S. M. Babamir, "A PSO-based task scheduling algorithm improved using a load-balancing technique for the cloud computing environment," *Concurrency Comput. Pract. Exper.*, vol. 30, no. 12, Jun. 2018, Art. no. e4368.
[11] C. Tang, M. Hao, X. Wei, and W. Chen, "Energy-aware task scheduling in mobile cloud computing," *Distrib. Parallel Databases*, vol. 36, no. 3, pp. 529–553, Sep. 2018.
[12] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu, "A multi-objective ant colony system algorithm for virtual machine placement in cloud computing," *J. Comput. Syst. Sci.*, vol. 79, no. 8, pp. 1230–1242, 2013.
[13] S. K. Mishra, D. Puthal, B. Sahoo, P. P. Jayaraman, S. Jun, A. Y. Zomaya, and R. Ranjan, "Energy-efficient VM-placement in cloud data center," *Sustain. Comput., Inform. Syst.*, vol. 20, pp. 48–55, Dec. 2018.
[14] K. Zhang, Y. Mao, S. Leng, A. Vinel, and Y. Zhang, "Delay constrained offloading for mobile edge computing in cloud-enabled vehicular networks," in *Proc. 8th IEEE Int Workshop Resilient Netw. Design Modeling (RNDM)*, Sep. 2016, pp. 288–294.
[15] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, and L. Pan, "Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.
[16] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 1, no. 2, pp. 89–103, Jun. 2015.

[17] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.
[18] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 668–682, Mar. 2019.
[19] C. Dong and W. Wen, "Joint optimization for task offloading in edge computing: An evolutionary game approach," *Sensors*, vol. 19, no. 3, p. 740, Feb. 2019.
[20] T. Zhao, S. Zhou, X. Guo, Y. Zhao, and Z. Niu, "A cooperative scheduling scheme of local cloud and Internet cloud for delay-aware mobile cloud computing," in *Proc. IEEE Globecom Workshops*, Dec. 2015, pp. 1–6.
[21] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya, "Computation offloading for service workflow in mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, pp. 3317–3329, Dec. 2015.
[22] J. Xu, L. Chen, and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge computing," *IEEE Trans. Cogn. Netw.*, vol. 3, no. 3, pp. 361–373, Sep. 2017.
[23] I. Ketykó, L. Kecskés, C. Nemes, and L. Farkas, "Multi-user computation offloading as multiple knapsack problem for 5G mobile edge computing," in *Proc. IEEE Eur. Conf. Netw. Commun. (EuCNC)*., Jun. 2016, pp. 225–229.
[24] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.

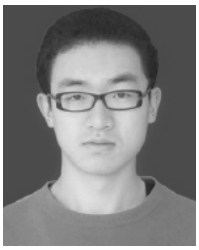
YUNMENG DONG received the M.S. degree in software engineering from Jilin University, China, in 2015, where he is currently pursuing the Ph.D. degree. His current research interests include distributed systems, cloud computing, edge computing, big data and data mining, and machine learning.



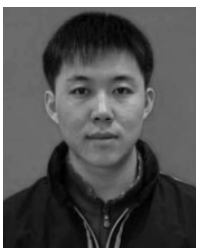
GAOCHAO XU received the B.S., M.S., and Ph.D. degrees from the College of Computer Science and Technology, Jilin University, China, in 1988, 1991, and 1995, respectively, where he is currently a Professor and a Ph.D. Supervisor. His main research interests include distributed systems, grid computing, cloud computing, the Internet of Things, information security, software testing, and software reliability assessment. As a person in charge or a principal participant, he has finished more than ten national-, provincial-, and ministerial-level research projects of China.



YAN DING received the bachelor's degree in computer science from Jilin University, in 2011, and the master's and Ph.D. degrees from the College of Computer Science and Technology, Jilin University, in 2014 and 2018, respectively. He is currently with the School of Computer Technology and Engineering, Changchun Institute of Technology, China. His main research interests include distributed systems, cloud computing, mobile cloud computing, the Internet of Things, virtualization technology, and big data. He has published several journal and conference papers in these areas.



XIANGYU MENG received the Ph.D. degree from the College of Computer Science and Technology, Jilin University, Changchun, China, in 2017. He is currently a Postdoctoral Researcher with Jilin University. His research interests include data mining, network security, cloud computing, and mobile edge computing.



JIA ZHAO received the Ph.D. degree from the College of Computer Science and Technology, Jilin University, in 2013. From 2016 to 2018, he was a Postdoctoral Researcher with the High-Assurance Software Laboratory, INEST TEC, and the University of Minho, Braga, Portugal. He is currently with the School of Computer Technology and Engineering, Changchun Institute of Technology, China. His main research interests include distributed systems, cloud computing, network technology, and big data. He has participated in several projects and published more than 30 journal and conference papers in these areas.