Taylor & Francis
Taylor & Francis Group

Automatika

Journal for Control, Measurement, Electronics, Computing and Communications

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/taut20

A novel edge server selection method based on combined genetic algorithm and simulated annealing algorithm

Yi-wen Zhang , Wen-ming Zhang , Kai Peng , Deng-cheng Yan & Qi-lin Wu

To cite this article: Yi-wen Zhang , Wen-ming Zhang , Kai Peng , Deng-cheng Yan & Qi-lin Wu (2021) A novel edge server selection method based on combined genetic algorithm and simulated annealing algorithm, Automatika, 62:1, 32-43, DOI: 10.1080/00051144.2020.1837499

To link to this article: https://doi.org/10.1080/00051144.2020.1837499

Published online: 02 Nov 2020.

AUTOMATIKA
2020, VOL. 62, NO. 1, 32–43
https://doi.org/10.1080/00051144.2020.1837499

A novel edge server selection method based on combined genetic algorithm and simulated annealing algorithm

Yi-wen Zhang[a], Wen-ming Zhang[a], Kai Peng[b], Deng-cheng Yan[c] and Qi-lin Wu[d]

[a]School of Computer Science and Technology, Anhui University, Hefei, People's Republic of China; [b]College of Engineering, Huaqiao University, Quanzhou, Fujian, People's Republic of China; [c]Institutes of Physical Science and Information Technology, Anhui University, Hefei, People's Republic of China; [d]School of Information Engineering, Chaohu University, Chaohu, Anhui, People's Republic of China

ABSTRACT
Mobile edge computing is a new paradigm which provides cloud computing capabilities at the edge of pervasive radio access networks in close proximity to users. The problem of edge server selection in mobile edge environment in terms of user's overhead is investigated in this paper. Due to the limited resources of edge server, we firstly study the task completion probability of edge servers. Secondly, we formally model the problem of edge server selection in terms of time latency and energy consumption. More especially, the computation overhead method for completing the task in cases of both service migration and non-migration is investigated. Then, a new optimized edge server selection algorithm, called combined Genetic algorithm and simulated Annealing algorithm for edge Server Selection (GASS) is designed. Finally, a series of experiments on a real-word data-trace are conducted to evaluate the performance of GASS. The results show that GASS can effectively minimize the overhead of the user and outperform traditional heuristic algorithms.

KEYWORDS
Mobile edge computing; edge server selection; time latency; energy consumption; service migration

1. Introduction

In recent years, with the rapid development of mobile devices, mobile applications such as natural language processing, facial recognition, and interactive online games have begun to appear and gained widespread attention [1–6]. Usually, these applications are time-sensitive, demand intensive computation and high-energy consumption [7]. Due to the physical size constraint, however, mobile devices are in general resource-constrained, having limited computation resources and limited battery life [8,9]. Therefore, how to solve the contradiction between the need of performing complex tasks and the limited resource in mobile devices becomes an important challenge.

To alleviate the resource scarcity of mobile devices, mobile cloud computing comes into being. Cloud computing centralizes computing and data resources, and then different users can get resources on demand [10,11]. However, users may experience long latency for data exchange with the public cloud through Wide Area Network(WAN) when connecting to centralized cloud data centres like Amazon EC2 [12–14] and Windows Azure [7]. Meanwhile, it is very difficult to reduce the latency in the wide area network.

Mobile Edge Computing (MEC) is envisioned as a promising approach to overcome these limitations [15–17]. As an extension of centralized cloud computing, MEC's core concept is to deploy edge servers to the edge of the network close to the users [7,18]. The edge server is usually collocated with the base stations or Wi-Fi access point. Thus the user can directly obtain the required resources from the edge server, which can reduce the latency caused by data transmission in WAN [19,20]. At the same time, it can reduce the communication burden and network congestion of the core network to a certain extent [21].

However, MEC still faces some challenges. On the one hand, due to the limited resources of each edge server and the open, dynamic and ever-changing features of the Internet [22,23], the edge server connected by the user may not have enough remaining resources to meet the user's request. Thereby, it is important for the mobile user to select the optimal edge server from candidate edge servers. On the other hand, each edge server covers a specific geographical area so that only the users within its coverage can connect to it [24]. Due to user's mobility, if the user leaves the coverage of the original edge server with unfinished task, to ensure the service continuity, another server should be selected to migrate the service running in the original server [25]. The reason is that if the mobile user continues to get resources from the original edge server, and transmit data to/from the user via other network nodes, other network nodes' communication resource may be occupied and the service response latency will increase [26]. Therefore, we should select the optimal edge server

CONTACT Kai Peng pkbupt@gmail.com

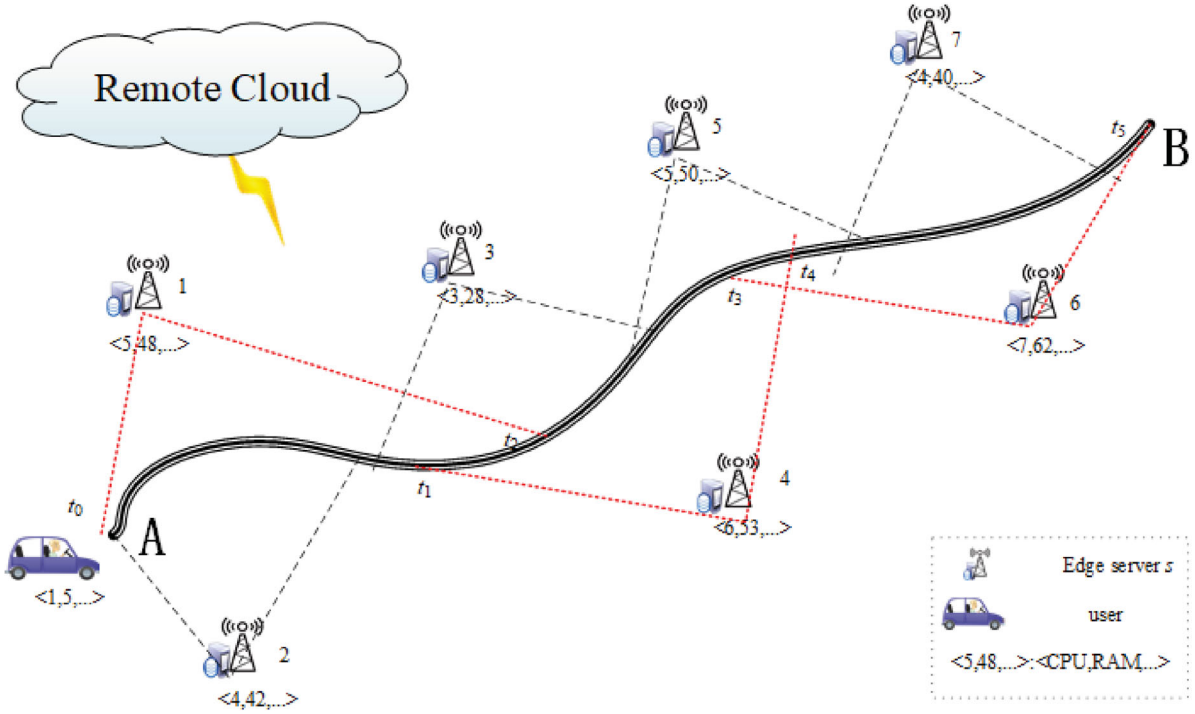

Figure 1. Application of selecting edge servers for mobile user in MEC.

to migrate the service when the service migration occurs.

To address these challenges, the edge servers should be select for the mobile user in advance. As illustrated in Figure 1, the mobile user is going to move from point *A* to point *B*. With the mobile user's moving, many edge servers need to select for the user to get resources. To improve the user's Quality-of-Experience (QoE) [27], edge server s_1, s_4, s_6 should be selected for the mobile user, because they have broader coverage and more resources comparing with other edge servers. In this paper, we study how to select edge servers for the mobile user to minimize the user's overhead. The following factors will affect the mobile user's overhead: (1) since service migration requires a large amount of data to be transferred between edge servers [26], frequent service migration can cause serious time latency; (2) in view of limited resources of each edge server, the edge server connected by user may be in overload state, so that the edge server will send the request to the remote cloud server, which will result in a additional long roundtrip latency [28]; and (3) user handheld mobile device has limited power and different energy consumptions are produced when the user connects different edge servers. Hence, it is necessary to propose an effective method to select the edge servers for mobile the user to minimize the overhead in terms of time consumption and energy consumption [29].

In order to solve the above problems, we propose the combined Genetic algorithm and simulated **A**nnealing algorithm for edge **S**erver **S**election (GASS) algorithm in this paper. And the proposed algorithm can select edge servers for the mobile user within polynomial time. The main contributions are as follows:

(1) We model the task completion probability of edge servers, and analyze the computation overhead of the user in cases of both service migration and non-migration;
(2) We propose GASS algorithm that selects edge servers in advance for user with known path, so as to minimize user's overhead in terms of time latency and energy consumption in mobile environment;
(3) Based on a real-world data-trace, we conduct extensive simulations to evaluate the performance of our GASS algorithm. The experimental results show that GASS can effectively reduce the overhead of the user compared with heuristic algorithms.

2. Related work

2.1. Edge server selection in static case

There have been extensive studies on the edge server selection in mobile edge computing. Tan et al. [30] proposed the first online algorithm for job dispatching and scheduling problem in edge-clouds with nontrivial competitive ratio. Firstly, users' tasks are assigned to appropriate edge servers, and then all tasks on the same edge server are scheduled to reduce the weighted response time of the tasks. Hamed et al [28] proposed hierarchical fog-cloud computing offloading and mod-

elled the competition among Internet of Things (IoT) users as a potential game to determine the computation offloading decisions. Besides, the authors also analyzed the properties of the formulated game and showed the existence of a pure Nash Equilibrium (NE). Phu et al. [24] further considered the coverage area of the edge servers and the resources of the edge server. The allocation problem of users was raised and modelled as a bin packing problem to minimize the number of hired edge servers while ensuring the required quality of service for users. These studies are very meaningful and focus much effort on reducing overhead. However, these approaches are only appropriate for computing user's overhead in static environment. If these approaches are applied in mobile environment, large deviations or fail will be resulted in. Different from them, we propose that the edge servers are selected for the mobile user in the mobile environment. At the same time, in order to ensure the continuity of the service, it is also necessary to take the service migration problem into account.

2.2. Service migration

In the traditional cloud migration decision model, the primary migration variable was the allocation of bandwidth resources [31] and the objective was to maximize the use of computing resources (such as CPU, memory, etc.). However, in the service migration process running in the edge cloud servers, user mobility is a key factor due to the limited coverage area of the edge server [32]. For that reason, the traditional cloud service migration decision model cannot directly be applied to the edge cloud server migration scenario. In [33], the authors studied the impact of user mobility on MEC performance, assuming that user mobility follows a random walk mobility model and used Markov chain to analyze service migration. A work on mobility-driven service migration based on Markov Decision Processes (MDPs) is given in [34], which mainly considered one-dimensional mobility patterns with a specifically defined cost function. And standard solution procedures are used to solve this MDP. Furthermore, a more effective method to solve this problem under the one-dimensional moving model is proposed in [10]. Its transmission overhead and migration overhead are set to a constant whenever it occurs. Compared with the one-dimensional user movement model, Pan et al. [15] proposed a more realistic two-dimensional user movement model to solve the large state space and approximated the underlying state space by defining the states as the distance between the user and the edge server location. Most of the above studies about service migration found the optimal threshold of service migration. It means that, when user leaves the coverage area of the original edge server, the user may make use of other network nodes as the relay node to get original edge server's resources without service migration. The following problems need to be taken into consideration. On the one hand, the bandwidth resource of other network nodes may be occupied; on the other hand, it may cause network congestion. In comparison, our edge server selection algorithm GASS requires that the mobile user directly connects to the edge server and the service is migrated if the user leaves the coverage of the edge server with unfinished task.

3. System model

In this section, we assume that the load of each edge server fits a normal distribution over a period of time and analyze the task completion probability of edge servers. Then, we compute the overhead of completion task in cases of both service migration and non-migration. Finally, we can obtain the user's total overhead during user's moving.

3.1. Probability computation model

When the edge server provides resource to the user, the server resource will be reduced correspondingly (in this paper, CPU resources represent server resources). Due to the limited resources of the edge server, there is no guarantee that the edge servers connected by the user can provide the required resources for the user. Therefore, it is necessary to calculate the probability that edge server successfully completes the task, denoted by p. Assuming that the load of each edge server fits a normal distribution over a period of time, we set the probability density function of the edge server load as

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2}} + b \tag{1}$$

Here x denotes the load of the edge server whose definition domain is $[0, 2\mu]$ and $f(x)$ is the probability when the load is x. Further, the value of μ is equal to half of the total resources of the edge server and b is a constant, only related to the total amount of resources. We define b as follows:

$$b = \frac{\int_{-\infty}^{0} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} + \int_{2\mu}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}}{2\mu} \tag{2}$$

If the resources required by the user are k, the probability that the edge server can satisfy the resource required by the mobile user is

$$p = \int_{0}^{2\mu-k} f(x) \tag{3}$$

Figure 2 shows the standard normal distribution image and shows how to convert the standard normal distribution into the probability density function that we need. Further, Figure 3 shows the probability density

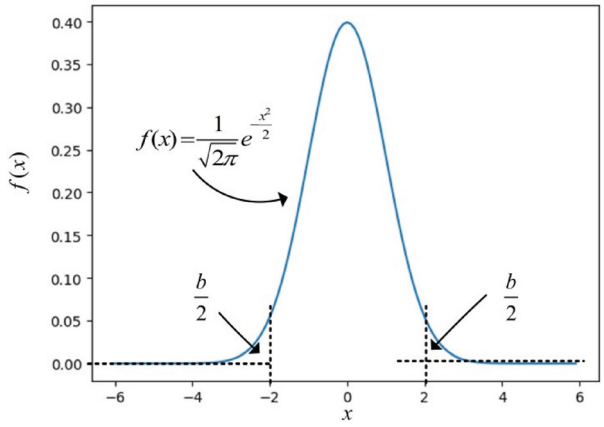

Figure 2. Normal distribution function.

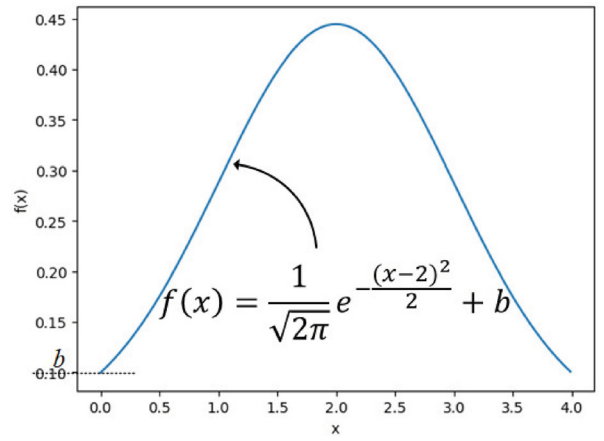

Figure 3. Probability density function.

function $f(x)$ of the edge server whose total resources is 4(i.e. $2\mu = 4$).

If the connected edge server cannot satisfy the resource requested by the user, the edge server will send the request to the remote cloud to get the resource required by the mobile user. The probability is 1–p. At any time, the user can connect to the remote cloud server, and the remote cloud server always is able to provide the resource required by the mobile user.

3.2. Computation model of user's overhead

We then introduce the computation model of user's overhead. We assume that there are S edge servers in the system, where $1, \ldots, S$ denotes the set of edge servers. The mobile user can connect to these edge servers to transmit data through the wireless channel during the mobile process [28]. We further denote the set of all servers available as $0, 1, \ldots, S$ when the user moves, where 0 represents the remote cloud server which has sufficient computing resources. Because the edge server has limited resources, when the edge server cannot provide enough resource to the user, the user can get resources from the remote cloud server. And the user can also directly connect to the remote cloud server to obtain resources. We consider a task $T_i \triangleq (B_i, D_i)$ that is one of many tasks during the movement and the task can be computed either on the edge server or the remote cloud server. Here, B_i denotes the size of computation input data (e.g. the program codes and input parameters) involving in the task T_i, and D_i denotes the total number of CPU cycles required to accomplish the computation task T_i. The user can apply the methods in [35,36] to obtain the information of B_i and D_i. We next discuss the computation overhead in terms of energy consumption and time consumption in case of both service migration and non-migration.

3.2.1. No service migration

If task T_i is executed on the edge server, $d_{i,s} < R_s$ must be guaranteed when the user connects to the edge server s, where $d_{i,s}$ indicates the distance between the user handheld mobile device and the edge server s, and R_s is the coverage radius of edge server s. We assume that the mobile user can only connect to one server at any time. According to [8], the time the user transfers data to the edge server s is $B_i / r_{i,s}$, for task T_i. Here, $r_{i,s}$ indicates the data transfer rate between the user and the edge server s, which can know in advance by the user. The computing power of the edge server s is represented by F_s. If the user maintains connection with s before the edge server s completes the task T_i, the execution time of task T_i on the edge server is D_i / F_s. Similar to many studies such as [29,37], the time latency that the edge server sends the computation outcome back to the user is neglected, due to the fact that for many applications(e.g. face recognition), the size of the computation outcome in general is much smaller than the size of computation input data. So the response time and energy consumption of the task T_i executed in the edge server s can be given as follows, respectively

$$T_{i,s} = \frac{B_i}{r_{i,s}} + \frac{D_i}{F_s} \tag{4}$$

$$E_{i,s} = \beta_{i,s} \frac{B_i}{r_{i,s}} \tag{5}$$

where $\beta_{i,s}$ is a constant, mainly related to the transfer rate between the user and the edge server [28].

According to (4), (5), we can compute the overhead in terms of processing time and energy as

$$C_i = \lambda_T \left(\frac{B_i}{r_{i,s}} + \frac{D_i}{F_s} \right) + \lambda_E \beta_{i,s} \frac{B_i}{r_{i,s}} \tag{6}$$

where λ_T and λ_E denote the weights of time consumption and energy consumption for the user respectively.

When $\lambda_E \in [0, 1]$, we set $\lambda_T \in (0, 1]$ to avoid the user experiencing long latency [16]. The setting of these parameters depends on the user and the type of application. For example, when the device is at a low battery state, the user would like to increase the weight of energy consumption in order to save more energy. For a

latency-sensitive application, the user may increase the weight of weight of time latency to reduce the latency.

When the user directly connects to the remote cloud, the user's waiting time contains the task upload and server execution time, as well roundtrip latency d between the mobile user and remote cloud [28]. Then the time consumption and energy consumption are:

$$T_{i,0} = \frac{B_i}{r_{i,0}} + \frac{D_i}{F_0} + d \tag{7}$$

$$E_{i,0} = \beta_{i,0}\frac{B_i}{r_{i,0}} \tag{8}$$

where $r_{i,0}$ indicates the data transmission rate between the user and remote cloud and F_0 indicates the computing power of the remote cloud server. Further, the overhead in terms of time latency and energy consumption of the task $T_i$executed in remote cloud server can be given as

$$C_i = \lambda_T\left(\frac{B_i}{r_{i,0}} + \frac{D_i}{F_0} + d\right) + \lambda_E\beta_{i,0}\frac{B_i}{r_{i,0}} \tag{9}$$

It is worth noting that edge servers cannot always satisfy user's resource requests because of their limited resources. When the remaining resources of the edge server cannot meet the requirements, the edge server needs to send the request to the remote cloud. According to the model of task completion probability, the success rate that the edge server s completes the task is p. Thus, the overhead computation function that the edge server completes the task $T_i$is

$$\begin{aligned} C_i = {} & p\left(\lambda_T\left(\frac{B_i}{r_{i,s}} + \frac{D_i}{F_s}\right) + \lambda_E\beta_{i,s}\frac{B_i}{r_{i,s}}\right) \\ & + (1-p)\left(\lambda_T\left(\frac{B_i}{r_{i,s}} + \frac{D_i}{F_0} + d\right) + \lambda_E\beta_{i,s}\frac{B_i}{r_{i,s}}\right) \end{aligned} \tag{10}$$

3.2.2. Service migration

In the real-world, users often move around. Due to the contradiction between the limited coverage of single edge server and user's mobility, service migration should be realized to ensure the service continuity. With regard to service migration in MEC, the amount of data transferred (e.g. memory state data, application image data, input dataset, etc.) is always very large (e.g. in megabyte or gigabytes) [26]. To optimize the service downtime generated by service migration, the three-layer framework for migrating running service is used in this paper, which is transparent for users [38]. During the service migration process, the service downtime is related to transferring the runtime state (i.e. instance layer) and is only a fraction of the total migration time. In service non-migration scenario, the task is completed by one server. However, when the task is migrated, the task will be completed by different servers. We should consider not only the service downtime, but also the time it takes to complete the remaining task on another server. Therefore, the overhead computation model mentioned above can not be directly applied to service migration scenario. For ease of interpretation, in this paper we assume that the service downtime is a constant mt.

If the edge server s' is selected when the user leaves the coverage area of s with unfinished task, the execution time of the task $T_i$can be given as$(D_i - D'_i)/F_s + D'_i/F_{s'}$. Here, $D_i - D'_i$ represents the number of CPU cycles required to accomplish the part of task $T_i$on the server s and D'_i denotes the number of CPU cycles required to accomplish another part of task T_i on the edge server s'. Therefore, we can obtain the time consumption of accomplishing task T_i as

$$T_{i,s,s'} = \frac{B_i}{r_{i,s}} + \frac{D_i - D'_i}{F_s} + \frac{D'_i}{F_{s'}} + mt \tag{11}$$

Since the success rates that the edge server s and s' can satisfy the user resource request are p and p', the probability that both the edge server s and s' can satisfy the user's request is pp'; the probability that the edge server s can satisfy the user's request and the edge server s'cannot satisfy the user's request is $p(1–p')$. If the edge server s cannot satisfy the user resource request, the service migration is not performed later, and the remote cloud server completes the task T_i. The probability is $(1–p)$. We can compute the overhead function of service migration in terms of processing time and energy as

$$\begin{aligned} C_i = {} & pp'\left(\lambda_T\left(\frac{B_i}{r_{i,s}} + \frac{D_i - D'_i}{F_s} + \frac{D'_i}{F_{s'}} + mt\right)\right. \\ & \left. + \lambda_E\beta_{i,s}\frac{B_i}{r_{i,s}}\right) + (1-p)\left(\lambda_T\left(\frac{B_i}{r_{i,s}} + \frac{D_i}{F_0} + d\right)\right. \\ & \left. + \lambda_E\beta_{i,s}\frac{B_i}{r_{i,s}}\right) + p(1-p')\left(\lambda_T\left(\frac{B_i}{r_{i,s}} + \frac{D_i - D'_i}{F_s}\right.\right. \\ & \left.\left. + \frac{D'_i}{F_0} + d + mt\right) + \lambda_E\beta_{i,s}\frac{B_i}{r_{i,s}}\right) \end{aligned} \tag{12}$$

Consider another scenario. If the task T_i is not completed when the user leaves the coverage area of the edge server s and no candidate edge server is available for service migration, the task has to be migrated to the remote cloud server. In this case, the user's overhead function for the task T_i is

$$\begin{aligned} C_i = {} & p\left(\lambda_T\left(\frac{B_i}{r_{i,s}} + \frac{D_i - D'_i}{F_s} + \frac{D'_i}{F_0} + d + mt\right)\right. \\ & \left. + \lambda_E\beta_{i,s}\frac{B_i}{r_{i,s}}\right) + (1-p)\left(\lambda_T\left(\frac{B_i}{r_{i,s}} + \frac{D_i}{F_0} + d\right)\right. \\ & \left. + \lambda_E\beta_{i,s}\frac{B_i}{r_{i,s}}\right) \end{aligned} \tag{13}$$

It is also noted that if the user has already connected to the remote cloud, in order to reduce the consumption

caused by the service migration, no service migration is executed before the task is completed. Thereby, the overhead for completing task $T_i$can be obtained by (9).

3.2.3. Mobile user's total overhead

According to the above computation overhead model, when an edge server is selected, the computational overhead for completing a single task can be obtained. Usually, many tasks need to be completed for a user over a period of time. Due to user's mobility, the mobile user will connect to different edge servers to get resources. Next, we will show how the user selects the edge servers in the mobile path.

Definition 1 (Mobile Path). A mobile path is represented by a triple (*Time*, *L*, *F*), where:

(1) *Time* is the time span during which the user is moving. It includes a set of continuous time points;
(2) *L* is the set of the user's locations corresponding to all time points in *Time*;
(3) *F* is a mapping function between the time points and the user's locations on the path. F: $Time \rightarrow L$.

In the above definition, *L* divides the entire moving path into multiple segments. Function *F* represents the correspondence between time and location. Actually, it indicates the variable speed while a user moves. Specifically, if the time interval is small, it means the user moves with a high speed in that location; otherwise, the user moves slowly.

Definition 2 (Edge Server Selection). The selection of edge server during user movement is a triple (Se, S, G), where:

(1) *Se* represents the set of many segments, and $Path = se_1 \cup se_2 \cup \ldots$ (i.e. Combining all segments can compose the user's mobile path.);
(2) $M = 0, 1, \ldots, S$ represents the set of all the servers that the user can connect;
(3) *G* is a function representing the correspondence between the segment and all edge servers that cover the segment: $\forall se_i \in Se$, G: $se_i \rightarrow M$.

Since the user movement path is known in advance, according to the geographical location and the coverage radius of each edge server, we can get all candidate servers that the user can connect during the moving path: $0, 1, \ldots, S$. Function *G* divides the moving path into many segments *Se* and makes each segment be covered by the same severs. When the user moves to the position of the segment se_i, one of the candidate servers that cover the segment is select to provide resource for the mobile user.

Definition 3 (The Total Overhead). Assuming that the user's movement path is divided into *n* segments, all servers that the user can connect are represented as a vector of length *n*: $< s_1, s_2, \ldots, s_n >$, $s_i \in 0, 1, \ldots, S$. $\forall 1 \leq i \leq n-1$ and $s_i \neq s_{i+1}$, if a task is not completed before the user leaves the coverage area of the edge server s_i, the task needs to be migrated, otherwise it will not be migrated. Assuming that there are *m* tasks to be completed, the user's total overhead can be obtained according to the above calculation model: $\sum_{i=1}^{m} C_i$. Our goal is to minimize the total overhead of the mobile user: $min \sum_{i=1}^{m} C_i$

4. Edge server selection algorithm

In this section, we introduce the details about how to select edge servers for mobile user to minimize user's overhead. We divide the mobile path up into a set of segments and each segment is covered by the same edge servers. One of the candidate servers for each segment will be selected for the mobile user to get resources. From Figure 1, we can know that the quantity of edge servers that cover the user's mobile path will increase exponentially with the increasing length of moving path. Thereby, the number of edge servers that need to be selected will also increases. In this case, if we use enumeration method to select servers for mobile user, the minimal overhead can also be obtained. However, the method's complexity is $O(m^n)$, where *m* denotes the average number of candidate servers that the user can select in each segment and *n* denotes the number of segments of the user's moving path. So, as the scale of the problem increases, the enumeration method becomes impractical.

Therefore, we design the heuristic algorithm called GASS (combined Genetic algorithm and simulated Annealing algorithm for edge Server Selection). As we all know, the genetic algorithm and simulated annealing algorithm are two of the most widely used heuristic algorithms. The ability of genetic algorithm lies in powerful global searching. And the ability of annealing algorithm lies in avoiding being trapped in local optima. The GASS algorithm combines the advantages of the two algorithms, which introduces the temperature parameters of the simulated annealing algorithm into the genetic algorithm. GASS can inherit the powerful global searching ability of the genetic algorithm, decrease the converge speed to avoid being trapped in local optima in the early process and increase the convergence speed to improve efficiency in the later process [21]. The proposed algorithm can select edge servers for the mobile user to get an approximate optimal solution within polynomial time.1

Next, we introduce a more detailed combination of the genetic algorithm and simulated annealing algorithm and illustrate how GASS is applied to our edge server selection problem.

Step 1. In Table 1, the corresponding relationships between the GASS and edge server selection problem are shown. In the GASS algorithm, feasible

Table 1. Term matching between GASS and server selection problem.

GASS	Edge server selection
Chromosome	Selected servers
Gene	Server
Locus	Segment
Fitness	User's overhead in terms of time latency and energy consumption

solutions are modelled by chromosomes. Thus, the chromosome corresponds to the edge servers selected in each segment. Each chromosome comprises a set of independent genes, so we use genes to represent the selected servers. The locus of a gene in a chromosome expresses a segment during user movement. If a chromosome has a high fitness, it implies that the overhead of the selected servers is low.

Step 2. At the beginning of the initialization phase, we initialize the algorithm parameters including the population size, the initial temperature, terminating temperature, the cooling parameter, mutation rate, the iteration times and so on.

Step 3. Selection is a process of reserving the superior chromosomes and randomly weeding out a part of the inferior chromosomes. All the remaining chromosomes are regarded as the parent chromosomes.

Step 4. Crossover is an operation that recombines parent chromosomes to generate new child chromosomes. In a crossover process, a point is randomly chosen from the chromosomes firstly. After that, a randomly selected parent chromosome *parent*1 loses the genes after that point and another randomly selected parent chromosome *parent*2 loses the genes before that point, hence a new child chromosome *child* is generated by combining parent chromosomes. We define Δf as

$$\Delta f = f(child) - min(f(parent1), f(parent)) \tag{14}$$

where $f(child)$ represents the overhead of child chromosomes. If $\Delta f < 0$, it implies that the fitness of the child chromosome is better, then we add *child* to the population. Otherwise, we calculate the probability value $exp\ (-\Delta f\ /\ (k*T))$, where k is a constant and T is the current temperature. After that, a random number *rand* is generated in (0, 1). If $rand < exp\ (-\Delta f/(k*T))$, we add *child* to the population, otherwise, we discard *child*.

Step 5. Mutation is another operation used to generate new chromosome. In a mutation process, a single gene is randomly chosen from the parent chromosome and randomly changed to another feasible gene. We then get the new chromosome *child*. We define Δf as $\Delta f = f(child) - f(parent)$. If $\Delta f < 0$, we replace *parent* with *child*. Otherwise, we calculate the value $exp(-\Delta f/(k*T))$, a random number of *rand* is generated in (0,1). If $rand < exp(-\Delta f/(k*T))$, we replace *parent* with *child*, otherwise, reserve *parent* and discard *child*. The approach guarantees that chromosomes with higher fitness values have a higher probability to be reserved. In the earlier iterations, the aim is to decrease the converge speed and avoid the algorithm falling into local optima. In later iterations, the temperature becomes low, so that the likelihood of worse chromosomes replacing better ones becomes very low. Thereby, the convergence speed and the efficiency of the algorithm are improved [21].

The overhead of the chromosome is calculated according to the fitness function. GASS algorithm is executed iteratively and approximate optimal solution is eventually achieved. In the GASS algorithm, new individuals are generated through crossover and mutation. The GASS algorithm is summarized in Algorithm 1.1

The algorithm begins with initialization (Line 1). According to the user's moving path and the location of the edge server, a gene is randomly selected for each gene locus, and *count* chromosomes are generated to form a population *PopulationSet*. The selection operation is performed (Line 3), and a part of chromosomes with high fitness are directly retained according to the setting threshold. The chromosomes with low fitness are randomly reserved and the remaining chromosomes are directly discarded. Then, the steps crossover (Lines 5–9) and mutation (Lines 11–16) are processed, by which the chromosomes in the population are updated. Afterwards, the current optimal chromosome is recorded as *CurOptChr* (Line 17). Next, the current optimal chromosome is compared with the optimal chromosome that has ever recorded in the history, and the better one is recorded as *OptChr* (Lines 18–19). After that, cooling temperature (Line 20) is repeated until the lowest temperature is reached. Finally, *OptChr* is returned as the optimal chromosome (Line 21).

This proposed approach works well only when it knows the path of the mobile user and the position of edge servers in advance. An alternative way to get mobile users' paths is to make use of prediction methods utilized often in wireless mobile computing and communication [39]. In addition, when the user uses the navigation function, we can also get the user's moving path.

5. Simulated experiments and Analysis

In this section, we perform experiments to verify the performance of GASS and compare the results with other methods. All experiments were conducted on

```
Input parameters   initial temperature T, terminating temperature Tmin, cooling parameter α, quantity
                   of chromosomes count, …
Output result      The chromosome with the highest fitness
Randomly compose count chromosome in PopulationSet
While T > Tmin
    Select ParentSet from PopulationSet population and set ChildrenSet ← ∅
    While len(ChildrenSet) + len(ParentSet)<len(PopulationSet)
        generate the child chromosome by crossover operation
        If f(child)< f(parent)
            ChildrenSet ← child ∪ childrenSet
        Else if rand < exp(-Δf / ( k*T ))
            ChildrenSet ← child ∪ childrenSet
    For i =1 to len(PopulationSet)
        If rand < mutation_rate
            generate the child chromosome by mutation operation
            If f(child) < f(parent)
                parent ← child
            Else if rand < exp(-Δf / ( k*T ))
                parent ← child
    CurOptChr ← the chromosome with the best fitness
    If f(CurOptChr) < f(OptChr)
        OptChr ← CurOptChr
    T = T *α
Return OptChr
```

a Windows machine equipped with Inter Core i7 (3.6 GHz) and 16GB RAM. The algorithm is implemented by Python3.6.

5.1. Experiment settings

In our experiments, we adopt a hybrid dataset that is a mixture of the Shanghai Telecom [40,41] and Google Cluster [30]. The Shanghai Telecom dataset contains Internet information about service invocations on base station and we call base station as edge server in the following experiment. The mobile path can be obtained by selecting two points on the *Baidu map*, and planning a path between two points through the *Baidu map*. Therefore, we can get all the available edge servers that cover user's mobile path. The Google cluster data trace contains more than 120,000 jobs and each job contains a lot of tasks.

We assume that the user is equipped with two wireless interfaces, which are Long Term Evolution (LTE) and Wi-Fi [28]. The user may use LTE interface to communicate with the remote cloud server, while they can use the Wi-Fi interfaces to connect to nearby edge servers that can cover the user [42–44]. We assume that the energy parameter $\beta_{i,s}$ only depends on the type of interface and does not vary. According to [45], for the LTE interface, we set $\beta_0 = 2605$ mJ/sec. We further set $\beta_s = 1224.78$ mJ/sec [46], if the user connect to any edge server via Wi-Fi interface. The average transmission rate of LTE and Wi-Fi are 5.85 and 3.01 Mbps, respectively, as measured in [46]. According to these measurements, we assume that the transmission rate of Wi-Fi interfaces is randomly distributed over [2.01, 4.01] Mbps, and the transmission rate of LTE interface is set as 5.85Mbps. We also assume that the processing power of edge servers is randomly distributed in [2, 3] GHz and the processing power of remote cloud server is 4 GHz [47]. In this paper, the number of CPU of the edge server is randomly distributed over [4,7] and number of CPU of remote cloud server is very sufficient. For the convenience of calculation, the CPU number required by the mobile is set as 1. Unless otherwise stated, we assume a roundtrip latency of 200 milliseconds for remote cloud servers [30]. For the downtime generated by the service migration, we set a constant and assume that the service migrated belongs to high RAM APP, and the value is 15.3 s [38]. Similar to [8], we set $\lambda_T = 0.5$and $\lambda_E = 0.5$.

5.2. Experiment results

To verify the superiority of the proposed algorithm, we compare GASS algorithm with another three widely used methods: the genetic algorithm (GA), the simulated annealing algorithm (SA) and Greedy algorithm. In order to avoid the influence of iterations, the iteration number of all of the four methods is same. The tasks to be completed are randomly chosen from the data trace.

(1) *Impact of Iteration Times:* As shown in Algorithm 1, the parameter of iteration times is set in the initialization phase. This group of simulations is to verify the effectiveness of our algorithm and examines the impact of iteration times. During the experiment, the number of iterations of the algorithm is changed while the other parameters of the algorithm are kept unchanged. The number of iterations of the algorithm is related to the settings of the initial temperature T, the termination temperature $Tmin$, and the cooling rate α. When $T = 100$, $Tmin = 2$ and $\alpha = 0.98$,

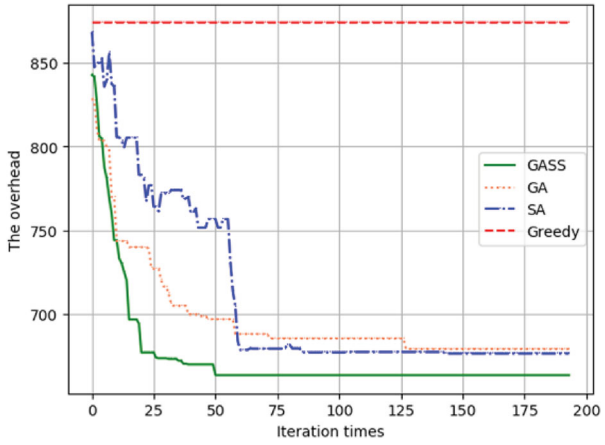

Figure 4. Impact of iteration times.

we set the radius of the edge server as 500 m, and the user's moving speed is kept at a constant speed of 6 m/s.

The impact of the iteration times is shown in Figure 4. In this experiment, our GASS algorithm outperforms the other three algorithms. With the increasing of iteration times, the overheads generated by GASS algorithm, SA algorithm and GA algorithm decrease stably and then converges to a stable value. However, The Greedy algorithm only selects the edge servers with the most resources, and does not consider the impact of user mobility, service migration, data transmission rate and other factors so that it is not capable of achieving the objective of minimal overhead. GA and SA algorithm can get reasonable results but compared to the GASS algorithm, they are inferior.

(2) *Impact of Edge Server Coverage Radius:* The impact of increasing edge server coverage radius on the experimental results is demonstrated in Figure 5, when the coverage of each server increases from 500metres to 1200 metres. The moving speed of the user keeps 6m/s. All experiments are repeated 100 times, and we use the average value as the result.

The impact of edge server radius on the number of candidate edge servers is shown in Figure 5 (a). It indicates that as the radius of the edge server service increases, the number of candidate edge servers that the user can connect increases. This is because that the larger coverage radius of the edge server, the larger area the edge server can cover. Thus, there are more edge servers that can cover the user's mobile path.

As shown in Figure 5(b), it can see that with the increasing of edge server's coverage radius r, the user's overheads generated by four algorithms decreases. The reason is that the larger radius of the edge server, the larger segment length that each edge server can cover.

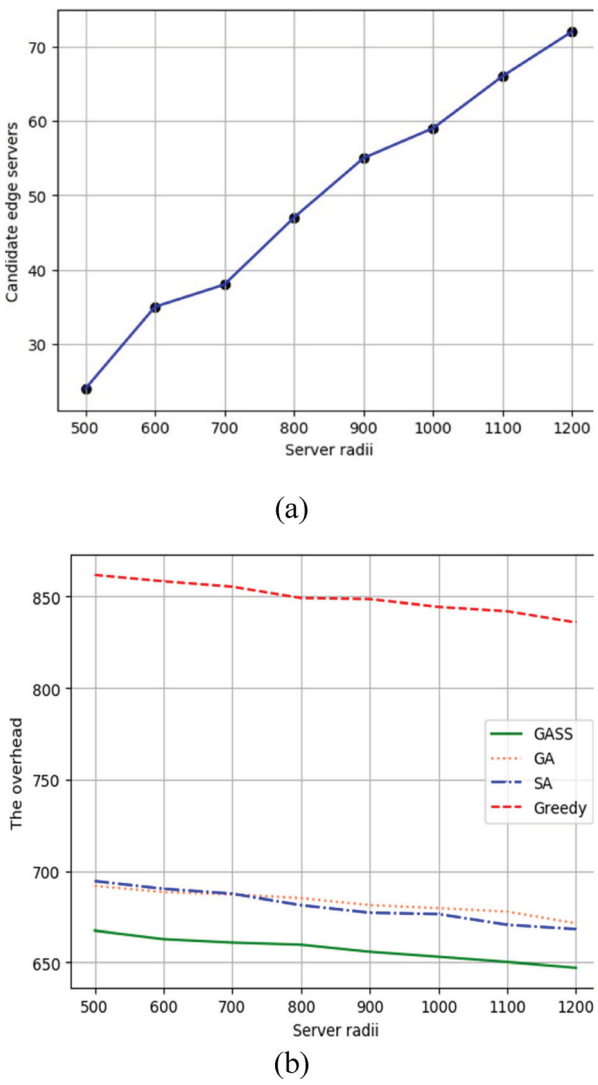

Figure 5. Impact of edge server coverage radius.

As a result, services may be migrated less frequently. Therefore, it may reduce the downtime of the service. Meanwhile, according to the result obtained from Figure 5(a), the user may have more choices when the user selects edge server in each segment so that the user can select the edge servers that produce less overhead. Meanwhile, from Figure 5(b), it can be seen that under the same conditions, the GASS algorithm is still superior to other three algorithms.

(3) *Impact of Mobile Speed:* the impact of the user's moving speed on the experimental results is shown in Figure 6. In this experiment, the edge server radius is set as 500 metres and the moving speed of the user increases from 6 m/s to 20 m/s. We implement the experiment for 100 times, calculate the overhead of the user and record the average value. The faster the user moves, the less time the user has passed the known path, and the fewer tasks the user completes on the mobile path.

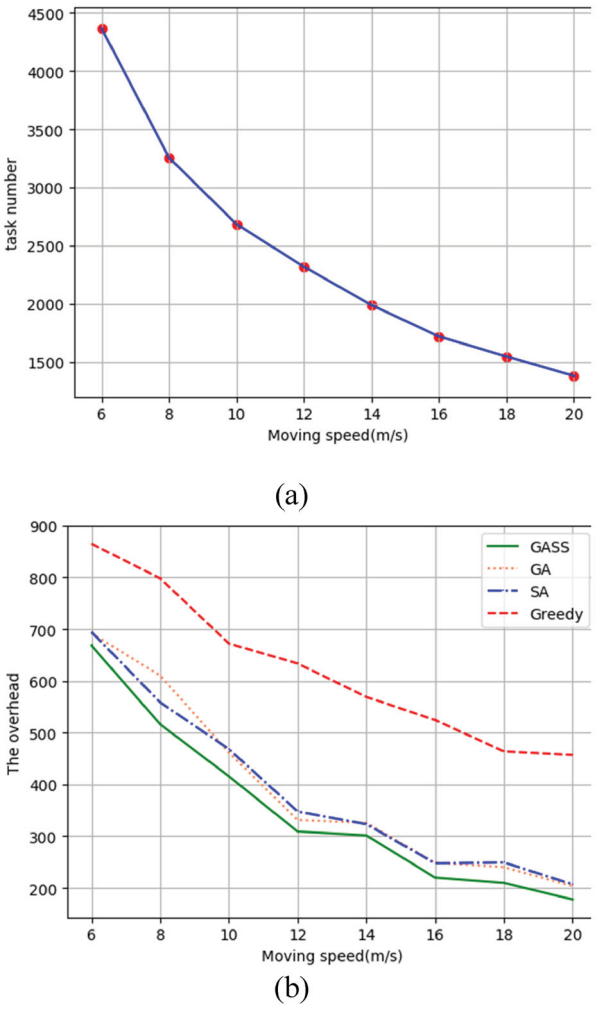

Figure 6. Impact of moving speed.

From Figure 6 (a), we can know that as the user moves faster, the number of tasks which need to be completed during the moving path also decreases. Thus, as can be seen from Figure 6 (b), the overhead of the user reduces with the increasing of moving speed. Meanwhile, the experimental results show that under the same conditions, the overhead of GASS algorithm is obviously less than that of GA algorithm and SA algorithm, and is better than that of greedy algorithm.

6. Conclusion

In this paper, we focus on the problem of edge server selection in mobile environment. We assume that the load of each edge server fits a normal distribution over a period of time. Thereby, the task completion probability of edge servers can be obtained. Furthermore, with the precondition that the task completion probability of edge servers is known, we can compute user's overhead in cases of both service migration and non-migration. To minimize user's overhead in terms of time consumption and energy consumption, we propose GASS algorithm that can select servers in advance for the user. The simulation experiments show that GASS outperforms traditional methods in terms of user's overhead. However, the application of the proposed method is restricted. The mobile path and the speed of all mobile users should be precisely predicted. Moreover, the proposed algorithm should be implemented as a service in cloud or edge cloud to obtain optimal result within a short time. In the future, we will also seek other algorithms to select edge server for the mobile user with higher effectiveness or efficiency.

Acknowledgements

This work is supported by National Natural Science Foundation of China (No. 61872002, No.61902133), Anhui Key Research and Development Plan (No. 201904a05020091), Natural Science Foundation of Anhui Province of China (No. 1808085MF197) and Natural Science Foundation of Fujian Province (No.2018J05106), the Fundamental Research Funds for the Central Universities (ZQN-817), Quanzhou Science and Technology Project under Grant (No. 2020C050R).

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This work was supported by Natural Science Foundation of Fujian Province: [grant number No.2018J05106]; Natural Science Foundation of Anhui Province of China: [grant number No. 1808085MF197]; National Natural Science Foundation of China: [grant number No. 61872002, No. 61902133]; Quanzhou Science and Technology Project : [grant number No. 2020C050R]; Anhui Key Research and Development Plan: [grant number No. 201904a05020091]; the Fundamental Research Funds for the Central Universities : [grant number ZQN-817].

References

[1] Soyata T, Muraleedharan R, Funai C, et al. Cloud-vision: real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In 2012 IEEE symposium on computers and communications (ISCC). IEEE; 2012 July. p. 000059–000066.

[2] Qi L, Yu J, Zhou Z. An invocation cost optimization method for web services in cloud environment. Sci Program. 2017;2017:4358536:1–4358536:9.

[3] Yin Y, Chen L, Xu Y, et al. Location-aware service recommendation with enhanced probabilistic matrix factorization. IEEE Access. 2018;6:62815–62825.

[4] Yin Y, Xu W, Xu Y, et al. Collaborative QoS prediction for mobile service with data filtering and Slope-One model. Mobile Information Systems. 2017;2017. DOI:10.1155/2017/7356213.

[5] Gao H, Duan Y, Miao H, et al. An approach to data consistency checking for the dynamic replacement of service process. IEEE Access. 2017;5:11700–11711.

[6] Gao H, Mao S, Huang W, et al. Applying probabilistic model checking to financial production risk evaluation

and control: A case study of Alibaba's Yu'e Bao. IEEE Trans Comput Social Systems. 2018;5(3):785–795.
[7] Ouyang T, Zhou Z, Chen X. Follow me at the edge: mobility-aware dynamic service placement for mobile edge computing. IEEE J Sel Areas Commun. 2018;36(10):2333–2345.
[8] Chen X. Decentralized computation offloading game for mobile cloud computing. IEEE Trans Parallel Distrib Syst. 2015;26(4):974–983.
[9] Peng K, Zhu M, Zhang Y, et al. An energy- and cost-aware computation offloading method for workflow applications in mobile edge computing. J Wireless Com Network. 2019; Article number 207. DOI:10.1186/s13638-019-1526-x.
[10] Wang S, Urgaonkar R, He T, et al. Mobility-induced service migration in mobile micro-clouds. 2014 IEEE Military Communications Conference, Baltimore, MD 2014. p. 835–840. DOI:10.1109/MILCOM.2014.145.
[11] Gao H, Huang W, Yang X, et al. Toward service selection for workflow reconfiguration: an interface-based computing solution. Future Gener Comput Syst. 2018;87:298–311.
[12] Pan J, McElhannon J. Future edge cloud and edge computing for internet of things applications. IEEE Internet Things J. 2018;5(1):439–449.
[13] Qi L, Zhang X, Dou W, et al. A distributed locality-sensitive hashing-based approach for cloud service recommendation from multi-source data. IEEE J Sel Areas Commun. 2017;35(11):2616–2624.
[14] Gao H, Miao H, Liu L, et al. Automated quantitative verification for service-based system design: a visualization transform tool perspective. Int J Software Engineer Knowledge Engineer. 2018;28(10):1369–1397.
[15] Wang S, Urgaonkar R, Zafer M, et al. Dynamic service migration in mobile edge-clouds. 2015 IFIP Networking Conference (IFIP Networking), Toulouse; 2015. p. 1–9. DOI:10.1109/IFIPNetworking.2015.7145316.
[16] Peng K, Leung VCM, Xu X, et al. A survey on mobile edge computing: focusing on service adoption and provision. Wirel Commun Mob Comput. 2018;2018:8267838:1–8267838:16.
[17] Xu X, Liu Q, Luo Y, et al. A computation offloading method over big data for IoT-enabled cloud-edge computing. Future Gener Comput Syst. 2019;95:522–533.
[18] Yin Y, Chen L, Xu Y, et al. Qos prediction for service recommendation with deep feature learning in edge computing environment. Mobile Netw Appl. 2020;25:391–401. DOI:10.1007/s11036-019-01241-7.
[19] Wu H, Deng S, Li W, et al. Request dispatching for minimizing service response time in edge cloud systems. 2018 27th International Conference on Computer Communication and Networks (ICCCN), Hangzhou; 2018. p. 1–9. DOI:10.1109/ICCCN.2018.8487354.
[20] Gao H, Zhang K, Yang J, et al. Applying improved particle swarm optimization for dynamic service composition focusing on quality of service evaluations under hybrid networks. IJDSN. 2018;14(2):1550147718761583.
[21] Wu H, Deng S, Li W, et al. Service selection for composition in mobile edge computing systems. In 2018 IEEE International Conference on Web Services (ICWS). IEEE. 2018 July. p. 355–358.
[22] Gao H, Huang W, Duan Y, et al. Research on cost-driven services composition in an uncertain environment. J Internet Technol. 2019;20(3):755–769.
[23] Gao H, Chu D, Duan Y, et al. Probabilistic model checking-based service selection method for business process modeling. Int J Software Engineer Knowledge Engineer. 2017;27(06):897–923.
[24] Lai P, He Q, Abdelrazek M, et al. Optimal edge user allocation in edge computing with variable sized vector bin packing. In International Conference on Service-Oriented Computing. Cham: Springer; 2018 November. p. 230–245.
[25] Xenakis D, Passas NI, Merakos LF, et al. Mobility management for femtocells in LTE-advanced: key aspects and survey of handover decision algorithms. IEEE Commun Surv Tut. 2014;16(1):64–91.
[26] Wang S, Xu J, Zhang N, et al. A survey on service migration in mobile edge computing. IEEE Access. 2018;6:23511–23528.
[27] Yin Y, Aihua S, Min G, et al. Qos prediction for web service recommendation with network location-aware neighbor selection. Int J Software Engineer Knowledge Engineer. 2016;26(04):611–632.
[28] Shah-Mansouri H, Wong VWS. Hierarchical fog-cloud computing for IoT systems: a computation offloading game. IEEE Internet Things J. 2018;5(4):3246–3257.
[29] Li T, Wu M, Zhao M, et al. An overhead-optimizing task scheduling strategy for Ad-hoc based mobile edge computing. IEEE Access. 2017;5:5609–5622.
[30] Tan H, Han Z, Li X-Y, et al. Online job dispatching and scheduling in edge-clouds. In IEEE INFOCOM 2017- IEEE Conference on Computer Communications. IEEE; 2017 May. p. 1–9.
[31] Guo T, Sharma U, Wood T, et al. Seagull: intelligent cloud bursting for enterprise applications. In Presented as part of the 2012 USENIX Annual Technical Conference (USENIXATC 12); 2012. p. 361–366.
[32] Zhang W, Hu Y, Zhang Y, et al. SEGUE: quality of service aware edge cloud service migration. In 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom). IEEE; 2016 December. p. 344–351.
[33] Taleb T, Ksentini A. An analytical model for follow me cloud. 2013 IEEE Global Communications Conference (GLOBECOM), Atlanta, GA; 2013. p. 1291–1296. DOI:10.1109/GLOCOM.2013.6831252.
[34] Ksentini A, Taleb T, Chen M. A Markov decision process-based service migration procedure for follow me cloud. 2014 IEEE International Conference on Communications (ICC), Sydney, NSW; 2014. p. 1350–1354. DOI:10.1109/ICC.2014.6883509.
[35] Cuervo E, Balasubramanian A, Cho D-K, et al. MAUI: making smartphones last longer with code offload. In Proceedings of the 8th international conference on Mobile systems, applications, and services; 2010 June. p. 49–62.
[36] Chun B-G, Ihm S, Maniatis P, et al. Clonecloud: elastic execution between mobile device and cloud. In Proceedings of the sixth conference on Computer systems; 2011 April. p. 301–314.
[37] Kumar K, Lu Y-H. Cloud computing for mobile users: can offloading computation save energy? IEEE Computer. 2010;43(4):51–56.
[38] Machen A, Wang S, Leung KK, et al. Migrating running applications across mobile edge clouds: poster. In Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking; 2016 October. p. 435–436.
[39] Deng S, Wu H, Tan W, et al. Mobile service selection for composition: an energy consumption perspective. IEEE Trans Autom Sci Eng. 2017;14(3):1478–1490.

[40] Wang S, Zhao Y, Huang L, et al. Qos prediction for service recommendations in mobile edge computing. J Parallel Distrib Comput. 2019;127:134–144.
[41] Wang S, Zhao Y, Xu J, et al. Edge server placement in mobile edge computing. J Parallel Distrib Comput. 2019;127:160–168.
[42] Yin Y, Xu Y, Xu W, et al. Collaborative service selection via ensemble learning in mixed mobile network environments. Entropy. 2017;19(7):358. DOI:10.3390/e19070358.
[43] Yin Y, Yu F, Xu Y, et al. Network location-aware service recommendation with random walk in cyber-physical systems. Sensors. 2017;17(9):2059.
[44] Zeng X, Xu G, Zheng X, et al. E-AUA: an efficient anonymous user authentication protocol for mobile IoT. IEEE Internet Things J. 2018;6(2):1506–1519.
[45] Kim Y, Kwak J, Chong S. Dual-side dynamic controls for cost minimization in mobile cloud computing systems. 2015 13th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt), Mumbai; 2015. p. 443–450. DOI:10.1109/WIOPT.2015.7151104.
[46] Kwak J, Kim Y, Lee J, et al. DREAM: dynamic resource and task allocation for energy minimization in mobile cloud systems. IEEE J Sel Areas Commun. 2015;33(12):2510–2523.
[47] Wang Y, Sheng M, Wang X, et al. Mobile-edge computing: partial computation offloading using dynamic voltage scaling. IEEE Trans Commun. 2016;64(10):4268–4282.