# Request Dispatching for Minimizing Service Response Time in Edge Cloud Systems


Hongyue Wu[1, 2], Shuiguang Deng[1], Wei Li[2], Samee U. Khan[3], Jianwei Yin[1], and Albert Y. Zomaya[2]

[1] College of Computer Science and Technology, Zhejiang University, Hangzhou, China
{hongyue_wu, dengsg, zjuyjw}@zju.edu.cn

[2] School of Information Technologies, The University of Sydney, Sydney, Australia
liwei@it.usyd.edu.au, albert.zomaya@sydney.edu.au

[3] Department of Electrical and Computer Engineering, North Dakota State University, Fargo, USA
samee.khan@ndsu.edu



***Abstract*—The emerging of mobile edge computing has significantly reduced the response time and Internet risk of service invocations. However, due to the distributed architecture and limited resources, balancing the load between edge servers to minimize the overall response time has become a critical objective for mobile edge computing. This problem is generally related to two aspects, request dispatching and service scheduling. To address this issue, we proposed a novel heuristic method called GASD (combined Genetic algorithm and simulated Annealing algorithm for Service request Dispatching). It tackles the problem by jointly conducting request dispatching and service scheduling. In addition, a solution combination algorithm is applied to reduce the computation complexity of the method. The experimental results show that the GASD method can achieve much lower overall response time than the compared methods. Moreover, the execution time of GASD is in a low order of magnitude and the algorithm performs excellent scalability as the experimental scale increases.**



***Keywords-mobile edge computing; service computing; response time; request dispatching; scheduling***


## I. INTRODUCTION

In recent years, mobile devices have become one of the fastest adopted consumer products of all time and highly integrated into our daily life [1]. Although mobile devices have been constantly improved in terms of computation, communication and storage, they are still resource constrained when facing applications demanding high computational resources, such as multimedia processing, social networking and natural language processing. To enable such applications to run effectively on mobile devices, mobile cloud computing (MCC) [2] is proposed to allow mobile devices to offload computationally intensive tasks to the resource-rich cloud service providers. By such means, mobile devices are reduced to I/O terminals, while servers execute the heavy work, and hence significantly extends the capability and storage of mobile devices. However, in such MCC systems, mobile devices connect directly to centralized Internet cloud center, which inherently incurs relatively long and unstable latency between the mobile devices and the cloud.

To solve this problem, mobile edge computing (MEC) is proposed, where an abstract edge tier is added between mobile devices and cloud servers. Edge servers could be fixed infrastructures connected to the wireless access networks, or augmented routers and switchers in the wireless access networks [3]. MEC brings the cloud closer to mobile users by caching the cloud services on edge servers and shifting service execution from remote cloud servers to relatively close edge servers. MEC not only has the advantages of MCC, but can further reduce the response time of service invocations. Moreover, with service parameters transmitted directly between edge servers and mobile devices instead of through the Internet, the response time and Internet risk of service invocation are significantly reduced.

However, MEC still faces some challenges. In contrast to powerful cloud servers, edge servers are resource constrained, so when an edge server is accommodated with excessive service requests, the execution of some requests will be delayed, which leads to the response time of these requests be extended, even exceeds the response time by invoking the services on remote cloud servers. Since in MEC systems, service consumers are generally moving, it is rather important to reduce the invocation delay of services as much as possible. This task can mainly be divided into two goals as follows:

*1) Selecting an appropriate edge server or cloud server for each service request:* For a service request, there may be more than one edge server can provide the required service. These edge servers are of different capabilities, resources, accommodated service requests, as well as varied data transmission time and service execution time. When edge servers are overloaded, excessive overall response time can be induced and the advantages of edge computing is diminished. In such circumstances, remote cloud server will become a better solution ascribing to its short execution time and waiting time of service invocations. Therefore, it is a pivotal problem to balance the request processing between edge servers to avoid overload or wasted resources in any edge server, and eventually minimize the overall response time of requests.

*2) Scheduling the accommodated service requests to minimize their average response time for each edge server:* For each edge server, as its resources are limited, service requests are generally waiting in queue for execution. These service requests are of different required resources and execution time. Therefore, proper scheduling is required to fully utilize the limited resources and thus minimize the overall response time of accommodated services.

In this paper, we focus on minimizing service response time by jointly considering request dispatching and request scheduling in edge cloud systems. A novel heuristic approach is proposed by combing the genetic algorithm and the simulated annealing algorithm to optimize the request dispatching process. The request scheduling of edge servers is addressed by a novel request scheduling algorithm. Then, we combined these two algorithms to minimize the overall response time of service requests. Moreover, we introduced a novel solution recombination strategy, which significantly improves the efficiency of the algorithms.

The rest of the paper is organized as follows. In section II, we discuss related works. In Section III, we define the problem and present the formal models. Then we describe the main operations and algorithms in detail in Section IV. In Section V, we show the evaluation experiments and analyze the results. Finally, we conclude the paper in Section VI.

## II. RELATED WORK

In mobile computing systems, reducing the response time of services is always a research hotspot. Multiple approaches have been proposed from different perspectives to address this problem. Many researchers tackle this problem through VM migration [4], [5], [6]. For example, in [4], Rodrigues et al. presented a hybrid method to minimize service delay in a scenario with two edge servers. Some studies pay attention to application partitioning [7], [8], [9], [10], which attempts to minimizing service delay by dividing service tasks into mutually independent components and executing them in parallel. Moreover, many works aim at lowering service response time by selecting services with the lowest response time for mobile users [11], [12], [13]. For example, in [11], to reduce the service delay, Deng et al. proposed a mobility model, a mobility-aware service response time computation rule, and a mobility-enabled service selection algorithm based on teaching-learning-based optimization.

Service scheduling has also has attracted extensive research efforts in mobile computing systems. Liu et al. proposed a delay-optimal computation task scheduling for mobile edge computing systems. By analyzing the average delay of each task and the average power consumption at the mobile device, they proposed an efficient one-dimensional search algorithm to find the optimal task scheduling policy [14]. To optimize operational costs while providing rigorous performance guarantees in edge cloud systems, Urgaonkar et al. developed a novel alternate methodology to solve this problem [15]. In [16], Wang et al. proposed a task scheduling approach that sets different processing priorities for different tasks based on the collected data on human health status to reduce its processing time. In our previous work, we have also proposed a lightweight service scheduling algorithm RESP, which can jointly manage service selection, scheduling and resource allocation to address the resource limitation issue in mobile cloud systems [1].

Effective request dispatching is a most common way to reduce service delay in mobile edge computing, which aims at balancing the workload among edge servers to avoid overload or wasted resources in any of the edge server. Yang et al. focused on the joint optimization of service placement and

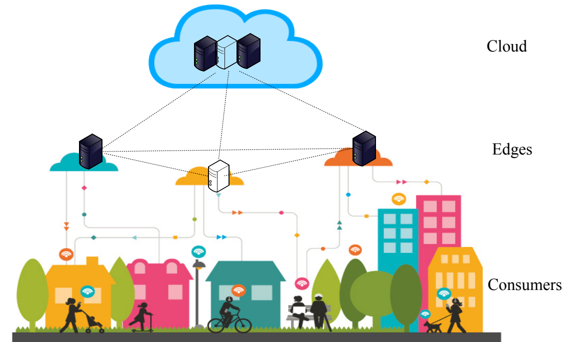


Figure 1. Mobile edge cloud architecture

load dispatching in the mobile edge cloud systems and developed a set of efficient algorithms for service providers to achieve various trade-offs among the average latency of mobile users' requests, and the cost of service providers [17]. To minimize the task completion time, Zeng et al. jointly considered the balancing of workload between client devices and computation servers, task image placement and balancing of the I/O interrupt requests among the storage servers and proposed a computation-efficient solution to solve it [18]. Jia et al. introduced a system model to capture the response times of offloaded tasks, and formulated a novel optimization problem, that is to find an optimal redirection of tasks between cloudlets such that the maximum of the average response times of tasks at cloudlets is minimized [19].

As mention above, both service scheduling and request dispatching have been widely addressed to lower service response time. However, most researches only consider one of the them and jointly considering them to reduce service delay has not been addressed so far to the best of our knowledge. As both of them are pivotal ways for service delay minimization, we jointly consider them in this paper.

## III. PROBLEM FORMULATION

### A. Edge Cloud Architecture

Our proposed framework is designed for edge computing systems, which are composed of three tiers, i.e. consumer tier, edge tier and cloud tier, as shown in Figure 1. We assume the following properties:

- There are one cloud center and multiple edge servers in the edge computing system. Edge servers can completely cover the whole 2-D area, so when a service request is proposed, it can be sent to a nearby edge server for processing.
- For each service request, only a part of edge servers can provide corresponding services to satisfy the request, while the cloud can provide all services.
- For any two edge servers, there exists a network path connecting them, and for each edge server, there exists a network path connecting it to the cloud.
- When an edge server receives a request, it can execute the service itself only if there is a corresponding service deployed on it, while if no corresponding service is deployed on it, it can only forward the

request to another edge server or the cloud for processing, and the result will be sent back after the service is completed and retransmitted to the user.
- The cloud is powerful enough to execute all incoming service requests simultaneously, which means that the services do not need to wait for execution in the cloud.

### B. Mobile Edge Computing Models

First, we give the formal definition of service requests. Service requests are proposed by consumers and sent to nearby edge servers via mobile networks.

**Definition 1 (Service Request).** A service request is represented as a 4-tuples $(id, t, e, E)$, where:

1) $id$ is the unique identifier of the request;
2) $t$ is the time when the request is received;
3) $e$ is the edge server where the request is received;
4) $E = \{e_j\}_{j=1}^{n}$ is the set of edge servers that can provide corresponding services for the request, and $n$ is the number of edges in the set.

As mentioned before, for each service request, only part of edges can provide services to satisfy it, so a request can only be dispatched to the cloud server or an edge server where a corresponding service is deployed.

**Definition 2 (Service).** A service is represented as a set of 5-tuples $(e, R, t, d_i, d_o)$ , where:

1) $e$ is the edge server or cloud server where the service is deployed;

2) $R$ describes the resources needed for $e$ to execute the service; $R$ can be denoted as a set of 2-tuples $R = \{(r_j, n_j)\}_{j=1}^{m}$, where $m$ is the number of types of required resources, and $r_j$ and $n_j$ denote the type and amount of the $j$-th kind of resource, respectively;

3) $t$ is the execution time, which is the makespan for $e$ to execute the service;

4) $d_i$ is the data size of the input parameters;

5) $d_o$ is the data size of the output parameters.

Each service is deployed on a specific edge. Requests and services are matched according to their functions. There is a one-to-many relationship between requests and services. That is to say, for a service request, there may be many services deployed on different edges can satisfy it, but one request corresponds to no more than one service in each edge server. Therefore, once a request is dispatched, the corresponding service is determined, so terms request and service will be interchangeably used without causing confusion thereafter in this paper.

**Definition 3 (Edge Server).** An edge server is represented as a 4-tuple $(S, A, I, L)$, where:

1) $S = \{s_1, s_2, \ldots\}$, describing the set of services that the edge server can provide;

2) $A$ is the available resources the edge server can utilize; it can be denoted as $A = \{(r_j, n_j)\}_{j=1}^{n}$, where $n$ is the total number of types of available resources and $r_j$ and $n_j$ denote the type and amount of the $j$-th kind of resource, respectively;

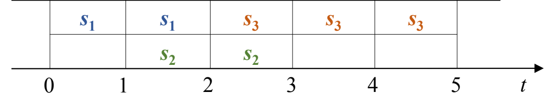


Figure 2. Execution sequence example

3) $I$ is a function used to describe the current idle resources of the edge server; Idle resources refer to resources that are available and not yet occupied by service executions; At a given time $t$, it can be represented by $I_t = \{(r_j, n_j)\}_{j=1}^{n}$;

4) $L$ is the execution sequence of the edge server, which will be introduced in detail in the following definition.

Each edge server manages an execution sequence, by which it conducts service execution, request insertion, scheduling, and resource allocation. In the following, we give the definition of execution sequence.

**Definition 4 (Execution Sequence).** For an edge $e$, its execution sequence describes the services that will be executed on each time unit. It can be formulated as a time function $L_t = \{s_j\}_{j=1}^{n}$, expressing that during time unit $t$, edge $e$ will execute $n$ services, $s_1, s_2, \ldots, s_n$, simultaneously.

Figure 2 shows an example of execution sequence, which shows that service $s_1$ will be executed from time point 0 to 2, service $s_2$ will be executed from point 1 to 3, and service $s_3$ will be executed from point 2 to 5.

According to Definitions 2-4, for each edge server, given a time point $t$, we can get the following resource equation:

$$\forall r,\ A(r) = I_t(r) + \sum_{s \in L_t} R_s(r) \tag{1}$$

where, $A(r)$ is the amount of available resource $r$ of the edge server, $I_t(r)$ is the amount of the idle resource $r$ at time $t$, and $R_s(r)$ is the amount of the required resource $r$ of service $s$. Equation (1) implies that, at each time point, the amount of overall available resources equals the summation of idle resources and resources occupied by service executions.

### C. Service Response Time Model

In this paper, we aim to minimize the time delay between the moment when an edge server receives a service request to the moment when it gets the result. The detailed definition is as follows.

**Definition 5 (Service Response Time).** The response time of a service is the expected delay between the moment when a request is received and the moment when the results are got by the edge sever. It can be calculated by

$$\mathcal{T} = t_i + t_w + t_e + t_o \tag{2}$$

where

1) $t_i$ is the input parameter transmission time, which can be calculated by

$$t_i = \begin{cases} \frac{d_i}{v_{jk}}, & if\ e_j \neq e_k \\ 0, & if\ e_j = e_k \end{cases} \tag{3}$$

where $d_i$ is the data size of input parameters, $e_j$ is the edge server where the request is received, $e_k$ is the edge or cloud server where the service is executed, and $v_{jk}$ is the data transmission speed between servers $e_j$ and $e_k$;

2) $t_w$ is the time delay for the service waiting to be executed in the edge server;

3) $t_e$ is the service execution time, which is the processing delay of the service;

4) $t_o$ is the output parameter transmission time, which can be calculated by

$$t_o = \begin{cases} \frac{d_o}{v_{jk}}, & if\ e_j \neq e_k \\ 0, & if\ e_j = e_k \end{cases} \quad (4)$$

where $d_o$ is the data size of output parameters.

It is no doubt that if a service is executed in the same edge server where the request is received, the input and output parameter transmission time can be saved.

## *D. Problem Statement*

**Definition 6 (Request dispatching problem).** Given current time $t$, the current states of all edges $\{e_j^t = (S_i, A_j, I_j^t, L_j^t)\}_{j=1}^{n}$, the data transmission speed between each two servers including all edge servers and the cloud server $\{v_{ij}\}_{i,j=1}^{n+1}$, and the incoming requests $q_1, q_2, q_3, \dots, q_m$, the request dispatching problem is to dispatch all the service requests to edge or cloud servers and schedule them to

$$\text{Minimize } \sum_{j=1}^{m} \mathcal{T}_j$$

s.t. $\forall j$,
$$s_j \in E_{q_j} \quad (5)$$

$$t + t_{q_j}^i \leq {}^\bullet s \quad (6)$$

$$s_j^\bullet - {}^\bullet s_j = t_{s_j}^e \quad (7)$$

$$\forall t' > t,\ \forall r,\quad \sum_{s \in L_j^{t'}} R_s(r) + I_j^{t'}(r) \leq A_j(r) \quad (8)$$

where $\mathcal{T}_j$ is the response time of $q_j$, $s_j$ is the selected service for $q_j$, $E_{q_j}$ is the set of edge servers for $q_j$, $t_{q_j}^i$ is the input parameter transmission time of $q_j$, $s_j^\bullet$ and ${}^\bullet s_j$ are the start time and end time of service $s_j$, respectively, and $t_{s_j}^e$ is the execution time of $s_j$.

As mentioned before, it is reasonable to regard minimizing response time as the optimization objective for request dispatching. In Definition 6, it is no doubt that each request should be dispatched to a server where a corresponding service is deployed, as shown by Equation (5). Equation (6) illustrates that services cannot start before the input data arrives. Equation (7) implies that the arrangement of each request is in accordance with its execution time. Moreover, the allocated resources should not exceed the available resources of the server at any time, as specified by Equation (8).

TABLE I. TERM MATCHING BETWEEN THE GENETIC ALGORITHM AND REQUEST DISPATCHING

| Genetic Algorithm | Request Dispatching |
|---|---|
| chromosome | Dispatching plan |
| Gene | Server |
| Locus | Service request |
| Fitness | Overall response time |

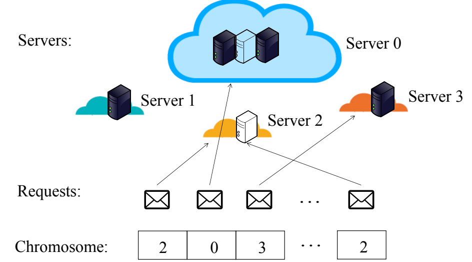


Figure 3. Genetic encoding scheme

It is not easy to solve this problem. In order to minimize the overall response time, we should jointly consider the global transmission delay, processing delay and waiting delay of all requests at the same time. Local edge or nearby edges are with lower transmission delay, but they are possibly not the best choice if the processing delay or waiting delay is high. In contrast, dispatching requests to remote edges or the cloud can cause high transmission delay, but the processing and waiting delay may be lower. Moreover, the dispatching of one request can affect the dispatching and scheduling of other requests, since our objective is to minimize the overall response time.

# IV. GASD APPROACH

In this section, we present our GASD approach. It is realized by combining and tailoring the genetic algorithm and simulated annealing algorithm for the proposed service request dispatching problem.

## *A. GASD Method*

The genetic algorithm [20] and simulated annealing algorithm [21] are two most widely used heuristic algorithms. The genetic algorithm has powerful global optimizing ability, but it can easily be trapped into local optima and its efficiency is not high. The simulated annealing algorithm is with relatively reasonable selection strategy, but its global optimizing ability is rather low. Therefore, we combine the advantages of the two algorithms by introducing the temperature parameters of simulated annealing algorithm into the genetic algorithm. The new algorithm can inherit the powerful global searching ability of the genetic algorithm, decrease the converge speed to avoid being trapped in local optima in the initial phase, and improve the convergence speed to improve efficiency in the end phase.

Table I presents the corresponding relationships between the parameters of the genetic algorithm and our request dispatching problem. Figure 3 shows an example of genetic

encoding. In the following, we introduce the operations in detail.

*1) Evaluation and Scheduling*

Evaluation is to evaluate the fitness of chromosomes. Fitness describes how well an individual fits the environment. In our model, chromosomes with high fitness correspond to dispatching plans with short overall service response time.

As mentation before, the response time of service requests is not only determined by the dispatching of service requests, but also significantly affected by the scheduling of services inside edge servers. Therefore, given a dispatching plan, in order to calculate the overall service response time, we should give the scheduling method of services inside edge servers. Reasonable request scheduling method is also important to minimize the response time of requests. Service scheduling is a rather difficult problem, as the scheduling of one request can affect the response time of other requests executed in the same edge server. In the following, we propose a scheduling algorithm to further minimize the global response time.

Given an incoming service request $q_0$ and the current state of the edge server $e$, request scheduling is to find the optimal start time of $q_0$ to minimize the average response time of all service requests executed in $e$. According to Definitions 5 and 6, we can get conclusion that minimizing the overall response time of all requests equals minimizing the start time of $q_0$ plus the summation of the response time gain of all other requests dispatched to $e$. Therefore, we can safely transform the objective of request scheduling as Equation (9):

$$Min\left(\mathcal{T}_{q_0}+\sum_{q\to e}\mathcal{T}_q\right) \Leftrightarrow Min\left({}^{\bullet}q_0+\sum_{q\to e}({}^{\bullet}q-{}^{\bullet}q')\right) \quad (9)$$

where $q \to e$ denotes that request $q$ is dispatched to $e$, and ${}^{\bullet}q$ and ${}^{\bullet}q'$ denote the start time of $q$ before and after the insertion of $q_0$, respectively. By this transformation, the complexity of the algorithm is significantly reduced, as the transmission delay and execution delay of the requests are eliminated.

Table II shows the detailed algorithm of request scheduling. For an incoming service request $q_0$, we first calculate its earliest start time $t_s$ (line 1) and then search the best insert time for $q_0$ (lines 3-17). The possible start time for $q_0$ is started from $t_s$ to the end of the execution sequence (line 3). For each time point $t$, we first remove all requests that start after $t$ (lines 4-6), then insert $q_0$ to its earliest possible start time (lines 7-10). The insert time should satisfy that, at each time point during the execution of $q_0$, each type of resource required by $q_0$ should be satisfied (line 8). After the insertion of $q_0$, we insert all transferred requests to its earliest possible start time (lines 11-15). If the current scheduling plan is better than the best plan, then it is set as the best plan (lines 16-17). Finally, the best solution is returned (line 18).

TABLE II. REQUEST SCHEDULING ALGORITHM ALGORITHM

| Input: | The incoming request $q_0=(id,t_0,e_0,E)$, the current state of the edge $\theta$: $(S,A,I,L)$, and the data transmission speed between $e$ and $e_0$ |
|---|---|
| Output: | The optimal scheduling plan |
| 1: | $t_s \leftarrow t_0+t^i_{q_0}$ |
| 2: | $\hat{\theta} \leftarrow \theta$ |
| 3: | **for** $t$= $t_s$ **to** $L^{\bullet}$ |
| 4: | **for** $q \in L_t$ |
| 5: | **if** ${}^{\bullet}q > t$ |
| 6: | remove $q$ to *List* |
| 7: | **for** $u$= $t$ **to** $L^{\bullet}$ |
| 8: | **if** $\forall t' \in (u, u+t^e_{q_0})$ and $\forall r \in R_{q_0}$, $R_{q_0}(r) < I_{t'}(r)$ |
| 9: | insert $q_0$ to $e$ s.t. ${}^{\bullet}q_0 = u$ |
| 10: | **break** |
| 11: | **for** $q \in$ *List* |
| 12: | **for** $u$= $t$ **to** $L^{\bullet}$ |
| 13: | **if** $\forall t' \in (u, u+t^e_q)$ and $\forall r \in R_q$, $R_q(r) < I_{t'}(r)$ |
| 14: | insert $q$ to $e$ s.t. ${}^{\bullet}q = u$ |
| 15: | **break** |
| 16: | **if** $\Theta(\theta) < \Theta(\hat{\theta})$ |
| 17: | $\hat{\theta} \leftarrow \theta$ |
| 18: | **return** $\hat{\theta}$ |

*2) Crossover and Mutation*

Crossover is an operation to recombine two parent chromosomes to generate two new child chromosomes. In a crossover process, a point is randomly chosen from the chromosome; then, the two parent chromosomes hold their genes before the point unchanged and exchange the genes after the point, thus generates two new child chromosomes. From the request dispatching perspective, crossover is to recombine two dispatching plans by interchanging corresponding objective servers to generate two new dispatching plans.

Mutation is another operation used to generate new populations. In a mutation process, a single gene is randomly chosen from the parent chromosome and randomly changed to another feasible gene. From the request dispatching perspective, mutation is to randomly choose a service request and change its objective server to another feasible sever to generate a new dispatching plan.

*3) Recombination*

Recombination is to recombine the service execution sequences of edge servers according new request dispatching plans generated by crossover and mutation operations. Table II presents the response time calculation algorithm. However, the time complexity of the algorithm is rather high. Therefore, we propose the recombination operation to recombine the service execution sequences of edge servers to reduce the time cost of our request dispatching algorithm.

Crossover generates new chromosomes by reserving the front parts of parent chromosomes unchanged and exchanging the later parts of parent chromosomes. Accordingly, for a specific edge, the front parts of service requests dispatched to it are remain unchanged and the later parts are exchanged. Therefore, we can recombine the scheduling of edge servers by simply exchanging the later dispatched requests. By this, the fitness of the new generated child chromosomes can be simply calculated without invoking the request scheduling algorithm.

Mutation generates new chromosomes by changing one gene of parent chromosomes. Accordingly, the object server of the corresponding request is changed from one server to another. Therefore, we can recombine the scheduling of edge servers by simply deleting the request from the old objective server and inserting it to the new objective server according to the mutation operation, thus the fitness of the new generated

chromosome can be simply calculated without invoking the request scheduling algorithm.

*4) Selection*

Selection performs the task to reserve superior chromosomes and eliminate inferior chromosomes. Given a parent chromosome $\theta$, if a child chromosome $\theta'$ is generated via crossover or mutation operation based on $\theta$, we define $\Delta(\theta)$ as

$$\Delta(\theta) = \Theta(\theta') - \Theta(\theta) \tag{10}$$

where $\Theta(\theta')$ and $\Theta(\theta)$ represents the fitness of the child and parent chromosome, respectively. Then the selection is performed as follow:

- If $\Delta(\theta) < 0$, it implies that the fitness of child chromosome is better, then we replace $\theta$ with $\theta'$ in the population;
- Otherwise, it implies that the fitness of parent chromosome is better, then we calculate the probability

$$p = exp - \frac{\Delta(\theta)}{K \cdot temp} \tag{11}$$

where $K$ is a constant and *temp* is the current temperature; after that, a random number rand is generated in (0, 1); if $p > rand$, we replace $\theta$ with $\theta'$ in the population, otherwise, we reserve $\theta$ and discard $\theta'$.

Equation (11) guarantees that chromosomes with higher fitness values have a higher probability of being selected. Therefore, a new solution can be reserved to the new generation if it is better than or close to the old one, making the new generation become better. Worse solutions are also possibly reserved to new generations, especially in earlier iterations when the temperature is high, which decreases the converge speed to avoid the algorithm being easily trapped into local optima. In later iterations, the temperature becomes low, so the possibility for worse chromosomes to replace better chromosomes becomes low, thus improves the convergence speed and the efficiency of the algorithm.

### B. Algorithm and Analysis

Given the initial parameters, the GASD algorithm executes iteratively to search for better solutions. In each iteration process, new solutions are generated by crossover and mutation operations; then, selection is conducted to reserve superior individuals and weed out inferior individuals. As thus, superior solutions are more likely transmitted to the next generations, making the next generations fit the environment better. The detailed algorithm is presented in Table III.

The algorithm begins with initialization (line 1), where initial chromosomes are randomly generated and put into the chromosome set $\Phi$, and a chromosome is randomly selected as the optimal chromosome (line 2). Then, the pivotal iteration steps are processed (lines 4-16). The iterations are controlled by the temperature (lines 3-4) and include operations of crossover (lines 6-9) and mutation (lines 10-12), through which more new chromosomes are generated (lines 8 and 11) and selected (lines 9 and 12). The fitness of all chromosomes in the chromosome set are compared and the current optimal one is recorded (line 13). Next, the current optimal chromosome is compared with the optimal chromosome that has ever emerged in the evolutionary history, and the better one is assign to $\hat{\theta}$ (lines 14-15). After each iteration, the temperature is lowered (line 16) and the iteration is repeated until the temperature is lower than the lower bound. Finally, $\hat{\theta}$ is returned as the optimal chromosome (line 17).

TABLE III. GASD ALGORITHM

| | |
|---|---|
| Input: | The incoming requests, the current situation of all servers with the speed among them, population size *pz*, crossover probability *cp*, mutation probability *mp*, initial temperature $T_{max}$, terminal temperature $T_{min}$, and cooling rate α. |
| Output: | An approximate optimal chromosome $\hat{\theta}$ |
| 1: | randomly compose *pz* chromosomes in $\Phi$ |
| 2: | $\theta$ =random($\Phi$); $\hat{\theta} \leftarrow \theta$ |
| 3: | *temp*=$T_{max}$ |
| 4: | **while** *temp*>$T_{min}$ |
| 5: |   **for** each $\theta$ *in* $\Phi$ |
| 6: |     **if** *random.p* <*cp* |
| 7: |       random select $\theta'$ from $\Phi$ |
| 8: |       $\theta_1, \theta_2$= crossover $(\theta, \theta')$ |
| 9: |       $\theta$= selection$(\theta, \theta_1)$; $\theta'$= selection$(\theta', \theta_2)$ |
| 10: |     **if** *random.p* <*mp* |
| 11: |       $\theta_1$= mutation $(\theta)$ |
| 12: |       $\theta$= selection$(\theta, \theta_1)$ |
| 13: |   $\theta$ ←best($\Phi$) |
| 14: |   **if** $\Theta(\theta) < \Theta(\hat{\theta})$ |
| 15: |     $\hat{\theta} \leftarrow \theta$ |
| 16: |   *temp*=*temp* ∗ α |
| 17: | **return** $\hat{\theta}$ |

Obviously, the time complexity of the algorithm is polynomial. Moreover, we can adjust the efficiency of the algorithm by adjusting the initialization parameters. The result may be better if one or more parameters among *pz*, *cp*, *mp*, *mt*, $T_{max}$ and α are increased, but it will cost more time to obtain the result. On the contrast, if these parameters are decreased, the efficiency will be improved, but the result may be suboptimal.

## V. EXPERIMENTS

We have implemented the algorithms in Python and our experiments are conducted on a MacBook Pro (macOS Sierra Version 10.12.5).

Since no standard experimental platforms and test data sets are available, we generated our experimental data in a synthetic way:

- *Service:* Each service is assigned three integers as the input data size, output data size and service execution time, respectively, which are referred to [22]. Each service requires 3 kinds of resources, and the required quantity of each kind of resource is randomly generated in (2, 5).
- *Service request:* We assume that the number of incoming service requests in each time unit is fixed. The received edge server is randomly selected for

TABLE VI. VARIABLE SETTINGS

| Set | Request number | Edge number |
|---|---|---|
| 1 | 5-25 | 10 |
| 2 | 20 | 5-25 |

each request. The amount of available edge servers for each request ranges from 2 to 8.

- *Edge server:* Each server has 3 kinds of resources and the available number of each resource ranges from 5 to 10. The amount of edge servers is fixed.
- *Speed:* The data transmission speed is randomly generated from 1 to 5 for each two edge servers and the speed between each edge server and the cloud is fixed as 1.

We focus on the following two variables in the experiments:

- *Request number:* Request number is the number of generated service requests in each time unit.
- *Edge number:* Edge number is the number of edge servers that can provide services for each service request.

As analyzed in Section IV, the effectiveness and efficiency of the algorithm are mainly related to these two variables. We set two sets of parameters, as shown in Table IV. In each set, one of the two parameters is varied and the other one remains fixed. All experiments are repeated 200 times, and we use the average value as the results.

### A. *Effectiveness of GASD*

In this subsection, we compare the overall response time of requests dispatched by GASD with other four widely used methods:

- *Access Point Scheduling (APS):* Access point scheduling method conducts request dispatching by simply selecting the edge server with the fastest data transmission speed.
- *The Genetic Algorithm (GA):* GA is used to verify the effectiveness of GASD by introducing temperature factors of the simulated annealing algorithm.
- *The Simulated Annealing Algorithm (SA):* SA is used to verify the effectiveness of the global searching ability of GASD.
- *The Ant Colony Algorithm (AC):* AC is a widely used heuristic algorithm, which generates new solutions by assigning pheromones and performing selection according to the pheromones.

First, we examine the impact of the number of generated service requests on the overall response time of requests dispatched by the five algorithms. To this end, we set the experimental parameters according to Set 1 in Table IV. The results are shown in Table V, from which we can see that the overall response time of all of the five methods are rising with the increasing of the service request number. As the other parameters are fixed, with the increasing of the request number, more requests will be dispatched to the edge servers, which increases the load of edge servers, therefore, the waiting time of services is increased, thus makes the response

TABLE IV. IMPACT OF REQUEST NUMBER ON OVERALL RESPONSE TIME

| **Methods** | **5** | **10** | **15** | **20** | **25** |
|---|---|---|---|---|---|
| GASD | 274 | 708 | 1309 | 2051 | 2989 |
| GA | 280 | 734 | 1354 | 2104 | 3083 |
| SA | 277 | 736 | 1425 | 2256 | 3389 |
| AC | 305 | 782 | 1465 | 2307 | 3440 |
| APS | 397 | 1402 | 3108 | 5122 | 8200 |

TABLE V. IMPACT OF EDGE NUMBER ON OVERALL RESPONSE TIME

| **Methods** | **5** | **10** | **15** | **20** | **25** |
|---|---|---|---|---|---|
| GASD | 3033 | 2051 | 1799 | 1686 | 1590 |
| GA | 3127 | 2104 | 1853 | 1754 | 1631 |
| SA | 3525 | 2256 | 1952 | 1797 | 1681 |
| AC | 3596 | 2307 | 2004 | 1855 | 1720 |
| APS | 9950 | 5122 | 3872 | 3345 | 2790 |

time of the requests increased. The difference between GASD and the other three heuristic algorithms becomes larger with the request number increasing, which demonstrates the superiority of our GASD method. In addition, the overall response time of service requests dispatched by APS arises more sharply, because it only considers the data transmission speed. By this, the experiment verifies the importance to consider other factors including the transmitted data size and the execution time of requests, the conditions and accommodated services of edge servers, etc.

To examine the impact of edge number on the result of request dispatching, we set the experimental parameters according to Set 2 in Table IV. The results are shown in Table VI, which shows that the overall response time of the requests is reducing with the increasing of the edge number. With the increasing of edge servers, there are more edge servers to process the service requests, which makes the waiting time of services shorter, therefore, the overall response time of service requests decreases. The difference between the results of all methods is also decreasing with the edge number increasing. Moreover, the comparison of the five methods is in accordance with the experiments before: the result of GASD is always the best, the result of APS is always the worst, and the gap between the result of APS and the other four methods is relatively high.

From these experiments, we can draw conclusion that our GASD method can achieve the best request dispatching over the five methods. With the resources of edge servers becoming relatively shorten, the superiority of GASD becomes more obvious. Besides, the APS algorithm performs worst, which demonstrates that the conditions of edge servers and other request-related parameters have great effect on the dispatching of service requests.

### B. *Effectiveness of Service Scheduling*

In this set of experiments, we verify the importance of service scheduling on the overall response time of service requests. To this end, we design a new method, which does

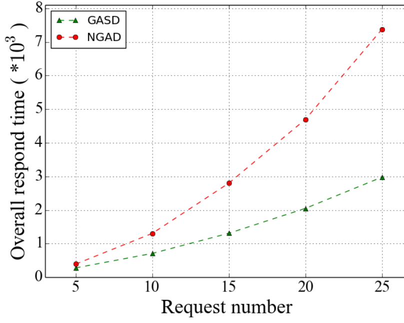


Figure 4. Impact of service scheduling ranging request number

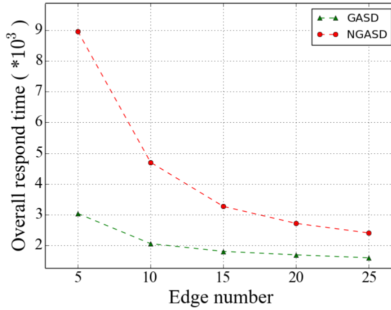


Figure 5. Impact of service scheduling ranging edge number

not account for scheduling of services in edge servers. This method processes service requests according to their arriving time. In the process of request dispatching, it adopts the GASD method to search for dispatching plans, without performing service scheduling. For an incoming service request, it simply find the earliest time when the idle resources are sufficient for the request and do not adjust the process sequence of the requests. We name this new GASD method as NGASD.

First, we examine the impact of request dispatching with the ranging of request number. To this end, we set the experimental parameters according to Set 1 in Table IV and implement GASD and NGASD methods. The result is shown in Figure 4. It shows that, with the request number increasing, the result of NGASD raises more sharply. This is because with the request number increasing, more service requests are dispatched to edge servers, which makes the waiting time of the services lengthened.

Then, we examine the impact of service scheduling with the range of edge number. To this end, we set the experimental parameters according to Set 2 in Table IV. The result is shown in Figure 5, from which we can see that, with the increasing of edge number, the overall response time of both methods are lowered, since the resources of edge servers are increased. Moreover, with the edge number increasing, the gap between the two methods narrows.

Therefore, we can draw conclusion that performing service scheduling to fully utilize the resources of edge servers performs a very important role in reducing the response time of service requests.

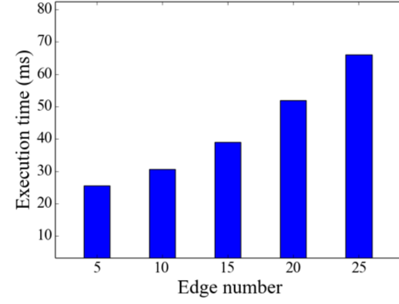


Figure 6. Impact of request number on the execution time of GASD

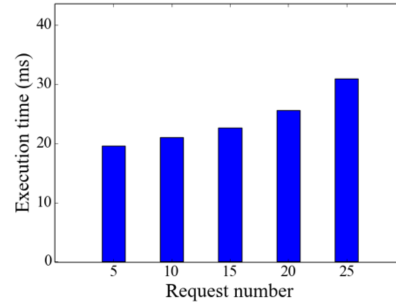


Figure 7. Impact of edge number on the execution time of GASD

### *C. Efficiency Evaluation*

In this subsection, we conduct two sets of experiments to examine the scalability of our method. Similarly, we also range those two parameters, i.e. request number and edge number.

Figure 6 shows the execution time of GASD with the ranging of request number, from which we can see that with the increasing of request number, the execution time of GASD method increases slightly. This is because the scale of the algorithm is affected by iteration numbers, which is determined by the temperature parameters in the initialization step. As the increasing of request number lengthens the chromosomes, which makes the time for crossover, mutation and selection operations become longer, there is a slight increase of the execution time of the algorithm, which is in accordance with the analysis in Subsection B of Section IV.

Figure 7 shows the execution time of GASD with the ranging of edge number, from which we can see that with the number of edge servers increasing, the execution time of GASD is raised slightly. This is because the increasing of edge servers makes the algorithm perform scheduling for more edge servers during the dispatching process. As the number of edge servers cannot affect the iterations of the algorithm, the execution time of the algorithm does not raise sharply.

Therefore, we can draw conclusion that the scale of the request dispatching problem only has a slight effect on the execution time of our GASD algorithm, which verifies the good scalability of GASD algorithm. Moreover, the execution time of the method is in a low order of magnitude, owing to

the great effort of the recombination strategy on reducing the computation complexity of the algorithm.

## VI. CONCLUSION

In this paper, we focus on service request dispatching problem in edge cloud systems. To further reduce the overall response time of service requests, we take service scheduling of edge servers into consideration. This problem is formally modeled, and a novel heuristic method is proposed by combining and tailoring the genetic algorithm and simulated annealing algorithm. A solution recombination strategy is proposed to improve the efficiency of the algorithm. In future, we will try to solve the problem with the movement of users taken into consideration. In that case, the request dispatching problem will become closer to the real world and the method will be more beneficial.

## ACKNOWLEDGMENT

This research was partially supported by Key Research and Development Project of Zhejiang Province (No. 2015C01027, No. 2015C01034, No. 2015C01029, and No. 2017C01013), Natural Science Foundation of Zhejiang Province (No. LY17F020014) and Major Science and Technology Innovation Project of Hangzhou (No. 20152011A03).

The work of Samee U. Khan was supported by (while serving at) the National Science Foundation. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

## REFERENCES

[1] H. Wu, S. Deng, W. Li, J. Yin, Q. Yang, Z. Wu, and A. Y. Zomaya, “Revenue-Driven Service Provisioning for Resource Sharing in Mobile Cloud Computing,” Proc. International Conference on Service-Oriented Computing, pp. 625-640, 2017.

[2] N. Fernando, S. Loke, and W. Rahayu, “Mobile cloud computing: A survey,” Future Generation Computer Systems, vol. 29, no. 1, pp. 84-106, 2013.

[3] L. Yang, J. Cao, G. Liang, and X. Han, “Cost aware service placement and load dispatching in mobile cloud systems,” IEEE Transactions on Computers, vol. 65, no. 5, pp. 1440-1452, 2016.

[4] T.G. Rodrigues, K. Suto, H. Nishiyama and N. Kato, ” Hybrid Method for Minimizing Service Delay in Edge Cloud Computing Through VM Migration and Transmission Power Control,” IEEE Transactions on Computers, 66(5), pp.810-819.

[5] L. Gkatzikis and I. Koutsopoulos, “Migrate or not? Exploiting dynamic task migration in mobile cloud computing systems,” IEEE Wireless Commun., vol. 20, no. 3, pp. 24–32, Jun. 2013.

[6] M. Mishra, A. Das, P. Kulkarni, and A. Sahoo, “Dynamic resource management using virtual machine migrations,” IEEE Commun. Mag., vol. 50, no. 9, pp. 34–40, Sep. 2012.

[7] Y. Wang, X. Lin, and M. Pedram, “A nested two stage game-based optimization framework in mobile cloud computing system,” in Proc. IEEE 7th Int. Symp. Service Oriented Syst. Eng., Mar. 2013, pp. 494–502.

[8] D. S. AbdElminaam, H. M. A. Kader, M. M. Hadhoud, and S. M. El-Sayed, “Elastic framework for augmenting the perfor- mance of mobile applications using cloud computing,” in Proc. 9th Int. Comput. Eng. Conf., Dec. 2013, pp. 134–141.

[9] S. Farrugia, “Mobile cloud computing techniques for extending computation and resources in mobile devices,” in Proc. 4th IEEE Int. Conf. Mobile Cloud Comput. Services Eng., pp. 1–10 , Mar. 2016

[10] X. Zhu, C. Chen, L. T. Yang, and Y. Xiang, “Angel: Agent-based scheduling for real-time tasks in virtualized clouds,” IEEE Trans. Comput., vol. 64, no. 12, pp. 3389–3403, Dec. 2015.

[11] S. Deng, L. Huang, D. Hu, J.L. Zhao and Z. Wu, ”Mobility-enabled service selection for composite services,”. IEEE Transactions on Services Computing, vol. 9, no.3, pp.394-407, 2016.

[12] S. Deng, L. Huang, J. Taheri, J. Yin, M. Zhou, and A.Y. Zomaya, ”Mobility-aware service composition in mobile communities,”. IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 47, no.3, pp.555-568, 2017.

[13] S. Deng, L. Huang, J. Taheri and A.Y. Zomaya, ”Computation offloading for service workflow in mobile cloud computing,” IEEE Transactions on Parallel and Distributed Systems, vol. 26, no. 12, pp. 3317-3329, 2015.

[14] J. Liu, Y. Mao, J. Zhang and K.B. Letaief,” Delay-optimal computation task scheduling for mobile-edge computing systems,”. In Information Theory (ISIT), International Symposium on, pp. 1451-1455, July, 2016.

[15] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan and K.K. Leung, ”Dynamic service migration and workload scheduling in edge-clouds,”. Performance Evaluation, 91, pp.205-228, 2015.

[16] H. Wang, J. Gong, Y. Zhuang, H. Shen, and J. Lach, ”Healthedge: Task Scheduling for Edge Computing with Health Emergency and Human Behavior Consideration in Smart Homes,”. In Networking, Architecture, and Storage (NAS), 2017 International Conference on (pp. 1-2). IEEE, Aug. 2017.

[17] L. Yang, J. Cao, G. Liang and X. Han, ”Cost aware service placement and load dispatching in mobile cloud systems,” IEEE Transactions on Computers, vol. 65, no.5, pp.1440-1452, 2016.

[18] D. Zeng, L. Gu, S. Guo, Z. Cheng and S. Yu, ”Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system,”. IEEE Transactions on Computers, vol. 65, no. 12, pp.3702-3712, 2016.

[19] M. Jia, W. Liang, Z. Xu and M. Huang, ”Cloudlet load balancing in wireless metropolitan area networks,”. In Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on, pp. 1-9, April, 2016.

[20] D. Goldberg, ”Genetic algorithms in search, optimization and machine learning,” Addison-Wesley Publishing Company, pp. 26-28, 1989.

[21] S. Kirkpatrick and M.P. Vecchi, ”Optimization by simmulated annealing. science,” vol. 220, no. 4598, pp. 671-680, 1983.

[22] Z. Zheng and M. R. Lyu,”QoS management of web services,”: Springer, 2013.