Priority Based Service Placement Strategy in Heterogeneous Mobile Edge Computing

Meiyan Teng[1], Xin Li[1,2,3(✉)], Xiaolin Qin[1], and Jie Wu[4]

[1] CCST, Nanjing University of Aeronautics and Astronautics
[2] State Key Laboratory for Novel Software Technology, Nanjing University
[3] Collaborative Innovation Center of Novel Software Technology and Industrialization
[4] Center for Networked Computing, Temple University
myteng@nuaa.edu.cn, lics@nuaa.edu.cn, qincs@nuaa.edu.cn, jiewu@temple.edu

Abstract. Mobile Edge Computing (MEC) is a promising method to reduce service delay by computational ability at the edge nodes. However, the limited resources at the edge nodes make it hard to response various services simultaneously. Hence, it is challenging to utilize the limited edge resources to host various service and reduce service response time. In this paper, we investigate the service placement problem such that the average service response time is minimized, which affects the user experience significantly. We define the priorities for nodes and services according to their contribution values, which indicates the influence for reducing service response time. Then, we propose a priority placement (2P) algorithm by taking both priority properties and local optimization into account. We conduct extensive simulations and the expeimental results show that the 2P algorithm can reduce the average service response time by $23\% - 46\%$, which indicates the 2P algorithm has better performance in reducing response time compared to the classical methods.

Keywords: Heterogeneity · MEC · service placement · low-latency · data-intensive.

1 Introduction

Nowadays, the scale and functions of the Internet of Things (IoT) have increased dramatically, heralding the arrival of the Internet of Everything. In the near future, the global IoT will expand to tens of billions of application devices [1]. Latency-sensitive application devices are on the rise, with devices such as augmented reality (AR) and driverless cars requiring real-time data processing. With the rapid development of IoT technology, traditional cloud is no longer suitable for IoT applications [2].

Mobile edge computing (MEC), which places servers at the edge of the network to offload cloud resources and reduce response delay, has drawn more and more attention [3]. It allows real-time analysis, testing, optimization on edge servers and sends data that needs to be processed centrally to the cloud server [4]. This technology not only reduces the burden on the cloud server, but

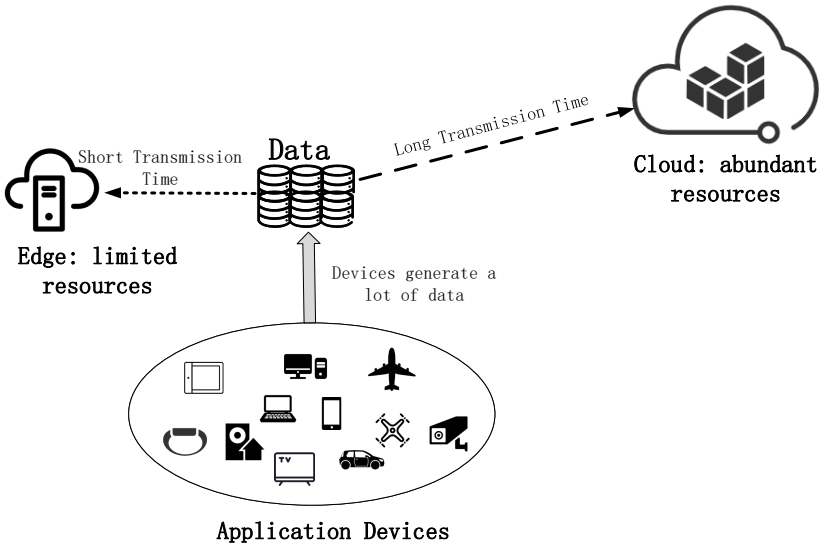


Fig. 1. Overview of cloud-edge in IoT environment.

also improves user experience. Edge computing is not a substitute for the cloud computing paradigm [5], but adds another layer of computing where it is close to the user.

However, a research institution predicts that by 2035, there will be 54 million driverless cars in the world [6]. Cameras on driverless vehicles that capture road conditions in real time generate about 1GB of data per second [7], even boeing 787 will produce 5GB data per second [8]. The service requests for such data-intensive applications not only put higher demands on latency, but also take up a lot of resources. Although the edge computing model has reduced the bandwidth pressure in network transmission and achieved low-latency response of services, the computing capacity and storage capacity of edge nodes are limited compared with those of cloud. This defect causes the services of numberous applications to be unresponsive and reduces user experience, as shown in Fig. 1. In this case, the problem of service placement needs to take both the cloud and edge nodes into account and makes a trade-off between them to achieve the goal of minimum response time.

In this paper, we investigate the services placement problem for service response delay reduction in a heterogeneous MEC system, as shown in Fig. 2. The cloud has sufficient resources to place all services, but the resources of the edge nodes are limited, which can communicate through the WAN. The user's requests are responded on the edge servers through the base station (BS) [9]. The heterogeneity of MEC implies that:

- the resource capacity of nodes and the requirements of services in the system are heterogeneous;

– the computational delays of nodes and the communication delays among them are heterogeneous.

In MEC system, the users' requests often change, and frequent service placement will generate a lot of energy consumption. For nodes that contain many service requests, their resources are constrained, but they have great value in improving user experience. In response to this situation, we define priorities for the nodes, and give priority to the placement strategy for the nodes with large contribution values. In addition, for nodes with tight resources, it is also a problem to consider which service to place first. So, we define service priorities based on the heterogeneous nature of services. And, we propose a priority placement (2P) algorithm to get a placement strategy with the lower system latency. The main contributions in this paper are as follows:

1) We make a detailed description of the research problem, construct a heterogeneous MEC system model, set system parameters, formulate the above characteristics and research problem.
2) We propose a priority placement (2P) algorithm to achieve services placement at MEC. The main idea is to set priorities based on service load distribution and set an upper limit on the number of identical service replicas to attain the goal of minimizing system response time.
3) We conduct extensive simulations, taking into account the priorities affected by load factors, as well as conducting comparative experiments. The results show that our algorithm has a significant improvement in reducing response delay under different parameters.

The rest of the paper is organized as follows. We summarize the related work in Section 2. Then, we describe the system model and formulize the service placement problem in Section 3, and propose our priority placement algorithm in Section 4. Next, we evaluate the performance of our algorithm in Section 5. Finally, we give conclusion remarks in Section 6.

2 Related Work

The research on service placement has attracted a lot of attention in recent years and there have been many research results. The most direct approach is to place the service on the local edge server, and [10] shows that this approach is feasible. In the case of adequate node resources, this strategy minimizes response time in the system, but if the resources are limited, most services will only be placed in the centralized cloud. Therefore, this strategy is obviously not the most efficient.

The authors in [11] study the scenario of overlapping node coverage and use random rounding technology to solve the service placement method under resource constraints. The service placement strategy studied in [12] and [13] fails to take the limitations of node capacity, computing power and transmission bandwidth into account, so the strict resource constraints of edge nodes cannot

be captured. In our paper, the scenario we studied is that node coverage does not overlap and resources are limited.

Xu *et al.* [14] weigh the delay and cost in the homogeneity MEC environment to obtain the placement strategy. He *et al.* [15] adopt greedy strategy, and the algorithm performance is better under the condition of homogeneous service performance. But the resource requirements for services are different and the performance is heterogeneous in general.

The authors in [16] adopt the way of node division to select the service with the greatest reward in a heterogeneous MEC system. This algorithm results in multiple placements of the same service replica on the uniform node. And frequent service placements increase system power consumption. And [17] adopts the linear programming method to jointly consider the service placement and request scheduling policies, but it is no work to consider the uneven load distribution of services in the case of heterogeneous MEC systems, which will have a great impact on the placement results.

In our paper, we define priorities for nodes and services based on the heterogeneous characteristics of nodes and services, then we propose a priority placement (2P) algorithm.

3 Problem Statement

3.1 Scenario and Notation

We focus on service placement issues within the heterogeneous MEC system, as shown in Fig. 2. The system is composed of some edge nodes and a remote centralized cloud. For the cloud, there are sufficient computing and storage resources to place services, which will generate faster computing speed. However, due to the distance from the data source and the limitation of communication bandwidth, the communication delay is high. So, it is suitable to place large services with insensitive latency. For edge nodes, they have limited resources to place services, which results in a longer calculation delay, but a lower communication delay. They are suitable to place small delay-sensitive services. The node characteristics and service requirements in MEC system are different. In this part, we use special symbols to represent the heterogeneous characteristics.

The set of nodes in the system is represented by $\mathbf{N}$, and the cloud is signed by N_0. For each node $n \in \mathbf{N}$ has a special performance $\langle R_n, \Gamma_n \rangle$.

- R_n represents the resource capacity of node n;
- Γ_n represents the communication delay time between node n and the cloud N_0.

The set of services in the system is represented by $\mathbf{S}$, and the performance attributes $\langle r_l, P_l \rangle$ of each service $l \in \mathbf{S}$ are different.

- r_l represents the resources that service l needs to consume when the service responds;
- P_l represents the number of replicas of service l in the system.

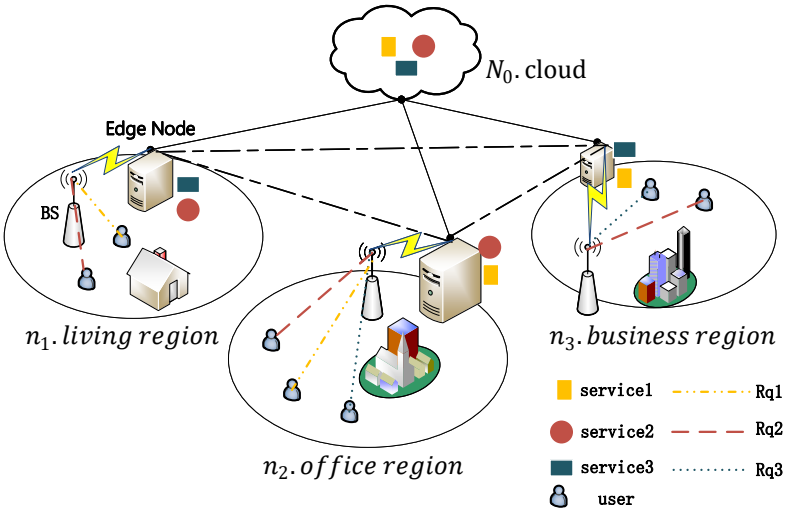


Fig. 2. System model.

In addition to the above characteristics, load is also an important factor. Node load refers to the number of users in the node. The service load refers to the number of service requests. The load of the nodes varies in different time periods, and the service load is unevenly distributed. For example, during working hours, there are more users in the office region than in living and business regions, and the number of requests for various types of services in the office region is also higher than in other regions. In other time periods, different phenomena will occur. In this article, we use $\boldsymbol{\Phi}$ to represent service load distribution.

– $\Phi_{l,n} \in \boldsymbol{\Phi}$ represents the load of service l within the coverage of node n.

Requests of services can be scheduled among edge nodes. For example, l_2 is not placed on n_3, so the users in this region send a request Rq_2 for l_2, which cannot be satisfied on the local server. The request is dispatched to node n_1 or n_2 in other regions through WAN. Based on the above characteristics, we set the total response delay as $\mathbf{T}$.

– $T^l_{m,n} \in \boldsymbol{T}$ represents the total response delay time when the request Rq_l within n region is scheduled to be served on node m.

The transmission time of the same request from n_1 to n_2 is the same as that from n_2 to n_1, but the computing power of n_1 and n_2 are different. To describe this characteristic of response time $\mathbf{T}$, we set

$$T^l_{m,n} = \partial^l_{m,n} + \beta^l_m \tag{1}$$

– $\partial^l_{m,n}$ represents the communication delay time when the request Rq_l within n region is scheduled to be served on node m, $\partial^l_{m,n} = \partial^l_{n,m}$, and $\partial^l_{n,n} = 0$;

– β_m^l represents the computing time of service l on node n.

According to the above description of the total response delay $\mathbf{T}$, we know that the response time of the service is related to which node the service responds to. We measure the efficiency of various algorithms by the response time of service requests. However, it is closely related to the service placement strategy.

3.2 Problem Formulation

We aim to minimize the service request response time according to a service placement strategy in a heterogeneous MEC system. So, we use vector $\mathbf{X}$ to represent the service placement scheme in the system. For $\forall l \in \mathbf{S}, m \in \mathbf{N}$,

$$x_{l,n} = \begin{cases} 1, & \text{service } l \text{ placed on the node n} \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

The response time of system request is represented by vector $\mathbf{Y}$, which is determined by placement scheme $\mathbf{X}$. We stipulate that if there are nodes that have placed service l, the requests for l will be scheduled to a node m with the shortest response delay. Otherwise, it will be scheduled to the cloud. It can be formalized as follows:

$$\begin{aligned} y_{l,n} =& \Theta\left(\sum_{m=0}^{|N|} x_{l,m} = 0\right) \times \Gamma_n + \Theta\left(\sum_{m=0}^{|N|} x_{l,m} \neq 0\right) \\ & \times \min\left\{T_{m,n}^l | x_{l,m} = 1\right\}, \quad m, n \in \mathbf{N}, l \in \mathbf{S} \end{aligned} \tag{3}$$

Where, if $\mathbf{E}$ is true, $\Theta(\mathbf{E}) := 1$; otherwise, $\Theta(\mathbf{E}) := 0$. $y_{l,n}$ represents the response time of the request for service l within the scope of node n.

The purpose of our research is to improve the quality of service. In this paper, our research goal is to minimize the response time of the system, which is expressed as follows:

$$\min \sum_{l=0}^{|S|} \sum_{n=0}^{|N|} y_{l,n} \tag{4}$$

$$\text{s.t.} \sum_{l=0}^{|s|} x_{l,n} \cdot r_l \leq R_n \quad \forall n \in \mathbf{N} \tag{4.1}$$

$$x_{l,n} \in \{0, 1\} \quad \forall l \in \mathbf{S}, \forall n \in \mathbf{N} \tag{4.2}$$

Where the constraint (4.1) means that the services placed on each edge node should not exceed its capacity.

From the above problem description, we learned that service placement is really about allocating limited node resources to services. We predict that service placement problem is an NP-hard problem [18]. Next, we will show the hardness of this problem.

Theorem 1. *The service placement in a heterogeneous MEC system is NP-hard.*

Proof: We will prove the theorem by a special case, which is constructed with the following assumptions: 1) We set the case with one cloud and one edge node, and 2) Only one replica is placed for each service. In this case, the best strategy is to place services on this edge node to lessen response time. However, the available resource in edge node is limited, so we need to choose some services to occupy the node resources such that the response delay is minimized. This problem can be inferred from the typical knapsack problem.

The typical knapsack problem can be formalized as follows. Given a set of items $A = \{a_i, 0 \leq i \leq n\}$, where the weight and the value of a_i is w_i and v_i, respectively. The problem is to select a subset A_s such that the total weight does not exceed the capacity W of knapsack and the total value is maximized. Then, in the scenario we constructed, the set of services is $S = \{s_l, 0 \leq l \leq n\}$, then for each service s_l, let the requirement of s_l be r_l, and the response time of s_l be y_l. We aim to select a subset S_s of services such that the total requirement does not exceed the resource capacity R of edge node and the response time is minimized.

If there is a strategy to select a subset A_s such that $\sum_{a_i \in A_s} w_i \leq W$, and $\sum_{a_i \in A_s} v_i$ is maximized. For each item $a_i \in A_s$, we can select a service $s_l \in S$ with $r_l = w_i$ and $\frac{1}{y_l} = v_i$. The services that are selected have the lowest response delay. In addition, if we select a subset S_s of S to minimize the total response time, we can get the subset A_s to maximize the value. Because the knapsack problem is NP-hard, we conclude that the services placement problem in MEC system is NP-hard.

□

4 Service Placement Strategy

To approximately minimize the response time of the service, we set priorities for all nodes and services according to the scene parameters, then determine the placement scheme according to the priority factors, so it is called priority placement (2P) algorithm. Firstly, the node with the highest priority is selected to study the placement strategy. Then we choose the service from the service candidate set of the node in order of the priority to place. In the process of service placement, we adjust the priority of nodes and services according to the deployed situation, make a dynamic priority definition. How to define node priority and service priority is a key point in the 2P algorithm. Next, we will introduce them one by one.

4.1 Total Delay Algorithm

The process of finding the total delay time in the system is described in Alg.1. Firstly, the input variables are defined. We set the attributes of nodes and

Algorithm 1 Total Delay Algorithm

Input: Node set $\mathbf{N}(n \in \mathbf{N})$, attributes: $\langle R_n, \Gamma_n \rangle$; Service set $\mathbf{S}(l \in \mathbf{S})$. attributes: $\langle r_l, P_l \rangle$; Schedule delay $\mathbf{T}$; Service load distribution $\boldsymbol{\Phi}$; Upper limit of replicas θ.

1: $\mathbf{Q} = averageDelay(\mathbf{T}, \boldsymbol{\Phi})$;
2: $\mathbf{G} = nodeCandidateSet(\mathbf{Q})$;
3: $\mathbf{L} = initServiceCandidateSet(\mathbf{G}, 1)$;
4: $\mathbf{X} = servicePlacement(\mathbf{L}, \mathbf{G}, \mathbf{Q}, r, R, \boldsymbol{\Phi}, \theta)$;
5: **for** each $l \in \mathbf{S}$ **do**
6: $P_l = getSum(\mathbf{X})$;
7: **if** $P_l == 0$ **then**
8: **for** each $n \in \mathbf{N}$ **do**
9: $y_{l,n} = \Gamma_n$;
10: **end for**
11: **else**
12: **for** each $n \in \mathbf{N}$ **do**
13: $y_{l,n} = minResponseTime(\mathbf{X}, \mathbf{T})$;
14: **end for**
15: **end if**
16: **end for**
17: **for** each $l \in \mathbf{S}$ **do**
18: **for** each $n \in \mathbf{N}$ **do**
19: $t = t + y_{l,n} \cdot \Phi_{l,n}$;
20: **end for**
21: **end for**

Output: Total delay time is t.

services as well as the upper limit of replicas θ. Through the network prediction, we can get the distribution of total response time and service load in a period of time, which is represented by the $\mathbf{T}$ and $\boldsymbol{\Phi}$, separately. According to the input variables, we obtain the node priority through the three-step method of lines 1-3 in Alg.1, which is introduced in detail in Section 4.2. Next, taking the variables obtained from lines 1-3 as input, line 4 adopts the *servicePlacement*() function to obtain the placement strategy $\mathbf{X}$; see Alg.2 for details.

Afterwards, based on placement scheme $\mathbf{X}$, lines 5-16 calculate the response time of the service request within the scope of each node. The number of replicas of each service in the system is calculated by the *getSum*() function. If $P_l = 0$, the system does not have a replica of service l, so the request of service l in each region must be responded to in the cloud, then $y_{l,n} = \Gamma_n$. If $\mathrm{P}_l \neq 0$, it indicates that there is at least one replica of this service in the system. Therefore, for requests in each region, we can define the *minResponseTime*() function on the basis of Eq.3 to get the minimum response time $y_{l,n}$. Finally, lines 17-21 calculate the total response time $\mathbf{Y}$ for all requests.

4.2 Node Priority

Uneven load distribution will cause a phenomenon that the more load of the node would lead to a higher value of the contribution to the research target,

but its resources are relatively tight. Our research goal is to minimize the total response time of all services in the system (e.g. Eq.4). Therefore, in order to better achieve the research goal, we properly defined the node priority after three steps of calculation.

Firstly, we assume that all requests of service l in the system are responded to node m. According to the input variables $\boldsymbol{\Phi}$ and $\mathbf{T}$, we can get the average response time of service l placed at node m, which is defined as follows:

$$Q_m^l = \frac{\sum_{n=0}^{|N|} T_{m,n}^l \cdot \Phi_{l,n}}{\sum_{n=0}^{|N|} \Phi_{l,n}}, l \in \mathbf{S}, m, n \in \mathbf{N} \tag{5}$$

In the same way, we calculate the value of other services on each node. In line 1 of Alg.1, we utilize the $averageDelay(\mathbf{T}, \boldsymbol{\Phi})$ function to implement Eq.5 and obtain matrix $\mathbf{Q}(Q_m^l \in \mathbf{Q}, l \in \mathbf{S}, m, n \in \mathbf{N})$, which is an important factor in the 2P algorithm.

Secondly, because the average delay time of each request Rq_l that was scheduled to diverse node is different $\left(Q_m^l \neq Q_n^l, l \in \mathbf{S}, m, n \in \mathbf{N}\right)$, we find the ideal node set of service according to the sequence of $\mathbf{Q}$, as follows:

$$G_l = \left\{ n_{p_1}, n_{p_2} \cdots n_{p_i} \cdots | Q_{n_{p_1}}^l \leq Q_{n_{p_2}}^l \leq \cdots \leq Q_{n_{p_i}}^l \leq \cdots, n_{p_i} \in \mathbf{N} \right\} \tag{6}$$

Where $G_{l,1} = n_{p_1}$ indicates that the first ideal node of service l is n_{p_1}, $G_{l,2} = n_{p_2}$ indicates that the second ideal node of service l is n_{p_2}, ..., and $G_{l,i} = n_{p_i}$ indicates that the i^{th} ideal node of service l is n_{p_i}.

In line 2 of Alg.1, we adopt the $nodeCandidateSet\,(\mathbf{Q})$ function to obtain the two-dimensional matrix $G_{\mathbf{S}\times|\mathbf{N}|}$, which represents the ideal node sequence for each service. The matrix $\mathbf{G}$ is composed of variables $G_{l,i} \in \mathbf{N}$, where $l \in \mathbf{S}, i \in |\mathbf{N}|$. Furthermore, we get a verdict from the two-dimensional matrix $\mathbf{G}$:

- row vector $G_{l,|\mathbf{N}|}$ shows the ideal node sequence of service l;
- column vector $G_{\mathbf{S},i}$ shows the i^{th} ideal node of all services.

Thirdly, we use function $initServiceCandidateSet()$ based on $\mathbf{G}$ to obtain the initial service candidates set $\mathbf{L}$ of each node, as shown in Alg.1 in line 3. It is defined as follows:

$$L_n = \left\{ l_{p_1}, l_{p_2}, \cdots l_{p_i} \cdots | G_{l_{p_i},1} = n, l_{p_i} \in \mathbf{S} \right\} \tag{7}$$

Initially, we put all services whose ideal node is n in the set L_n, and define them as the candidate service set of node n. The set L_n will be dynamically adjusted according to the placement situation. The more services in the set L_n, the higher the value of node n. So, the modulo $|L_n|$ is defined as the value of node n, also known as the priority.

4.3 Priority Placement Algorithm

Alg.2 describes the service priority placement function $servicePlacement()$ in detail. The main idea of the algorithm is to select services in order from the

Algorithm 2 Service Priority Placement Algorithm

Input: Average delay $\mathbf{Q}$; node candidate set $\mathbf{G}$; service candidate set $\mathbf{L}$; attributes of nodes $\langle R_n, \Gamma_n \rangle$; attributes of services $\langle r_l, P_l \rangle$; service load distribution $\boldsymbol{\Phi}$; Upper limit of replicas θ .

1: **while** (! isEmpty ($\mathbf{L}$)) **do**
2: $e \leftarrow \text{argmax}_{n \in N} |L_n|$;
3: Order $L_e = \{l_{p_1}, l_{p_2}, \cdots, l_{p_k}\}$,so that $\Omega_{l_{p_i}} \geq \Omega_{l_{p_{i+1}}}, \forall i < k$;
4: **for** each $l_{p_i} \in L_e$ **do**
5: **if** $r_{l_{p_i}} \leq R_e$ **then**
6: $x_{l_{p_i},e} = \mathbf{1}$;
7: $R_e \leftarrow R_e - r_{l_{p_i}}$;
8: $P_{l_{p_i}} + +$;
9: **else**
10: $x_{l_{p_i},e} = \mathbf{0}$;
11: **end if**
12: **if** $P_{l_{p_i}} < \theta$ **then**
13: $e' \leftarrow findNextNode\,(\mathbf{G}, l_{p_i}, e)$;
14: update$\big(\Omega_{l_{p_i}}\big)$;
15: $L_{e'} \leftarrow L_{e'} \cup l_{p_i}$;
16: **end if**
17: **end for**
18: $Clear(L_e)$;
19: **end while**

Output: Service placement strategy is $\mathbf{X}$.

service candidate set of the node e with the highest value (priority) for placement strategy, and then add the service to the set L'_e of its sub-ideal node e'.

Firstly, for the initial service candidate set $\mathbf{L}$ of each node, loop lines 1-18 until the service candidate set L_n of all nodes is null. In circulation, we select the node with the highest priority, which is marked as e, as shown in line 2 of Alg.2. Then we order the services in L_e based on service priority Ω. We consider the service loads, the number of replicas, the average response time and the response time gap with the sub-ideal node to define the variable Ω, as follows:

$$\Omega_{l,e} = \frac{\Delta Q_l + k_1 \cdot \sum_{n=0}^{|N|} \Phi_{l,n}}{Q_e^l + k_2 \cdot P_l}, l \in \mathbf{S}, e, n \in \mathbf{N} \tag{8}$$

Suppose that node e is the i^{th} ideal node of service l, $G_{l,i} = e$. So, Q_e^l is the average response time of all requests that are scheduled to node e. If $G_{l,i+1} = e'$, then $\Delta Q_l = Q_{e'}^l - Q_e^l$. In addition, $\sum_{n=0}^{|\mathbf{N}|} \Phi_{l,n}$ is the total load of the service l. P_l represents the number of replicas of service l. k_1 and k_2 is a parameter. In Eq.8, it is known that the time gap and service loads are greater, the service priority is higher. And the greater the average delay time and the number of replicas are, the lower the priority is.

Next, lines 5-11 implement placement in order of priority. If the remaining resources of the node meet the requirements of the service, place it, and make $x_{l,e} = 1$. Otherwise, $x_{l,e} = 0$. Then, we decide whether to continue the placement

according to the upper limit of service replicas θ on lines 12-16. If the number of service replicas in the system does not reach the upper limit θ, it will be added to the service candidate set $L_{e'}$ of the next ideal node e' that we use $findNextNode()$ function to find out. In this case, the priority of service and node e' will be adjusted dynamically. We update service priority $\Omega_{l,e'}$ and node priority $|L_{e'}|$ according to Eq.8. Finally, through the $clear()$ function shown in line 18, we clean up the services in the L_e set. Continue the placement process in loops 1-18 until all nodes have zero priority.

5 Evaluation

In this section, we evaluate our algorithm. We take the greedy algorithm as the baseline and conduct comparative experiments for two significant factors. The basic strategy of the greedy algorithms is to place the service with the shortest latency on each node.

A comparison algorithm studies load factors, which we call the non-loaded algorithm. The non-loaded algorithm does not take into account the number of service requests in each node, which changes the average delay variable and service priority variable as follows:

$$Q_e'^l = \frac{\sum_{n=0}^{|\mathbf{N}|} T_{\mathrm{e},n}^l}{|\mathbf{N}|} \tag{9}$$

$$\Omega_{l,e}' = \frac{\Delta Q_l'}{Q_e'^l + k \cdot P_l} \tag{10}$$

Another comparison algorithm studies the service priority factor, which we call unitary priority placement (u2P) algorithm. The u2P algorithm redefines the service priority in Eq.8 as opposed to the 2P algorithm, set

$$\Omega_{l,e}'' = Q_e^l \tag{11}$$

We have carried out a detailed simulation experiment and compared the experimental results.

5.1 Simulation Settings

There is only one cloud and N edge nodes in the simulation system, and the resource capacity of the nodes is heterogeneous. We use the unit time (*time slots*) to express the response delay time, and the communication delay between the nodes and the cloud is set to $\Gamma_n \in [50, 60]$. The scheduling delay matrix $\partial_{m,n}^l (m \neq n)$ is set to be within the range of [5, 15], which is a symmetric matrix. The response delay jitter is set to $\beta_m^l \in [1, 5]$.

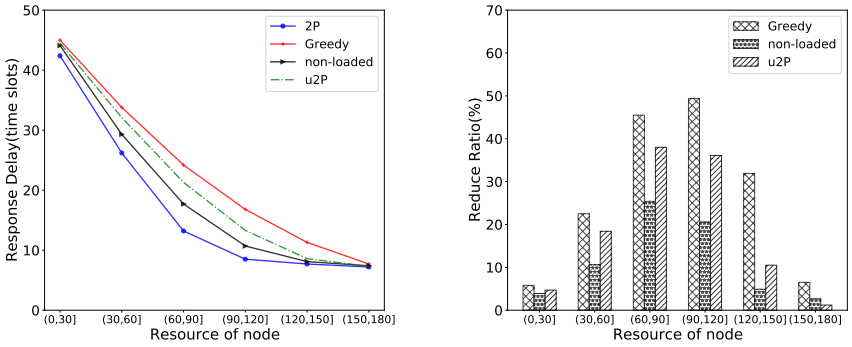
There are 100 services in the simulation system, and the amount of resource block occupied by each service is set to $r_l \in [2, 14]$. The loads of the services

within each node are distributed within [0, 100], which obeys the Gaussian distribution. We changed the three variables that are the number of nodes N, the resource capacity of nodes R_n and the upper limit of replicas θ. And we carried out three groups of comparative experiments.

In order to analyze our experimental results more clearly, we counted the average delay of each request, which is a measure to evaluate the superiority of the algorithms. At the same time, we calculate the reduction rate of the 2P algorithm in average response time relative to other algorithms. Supposing that the average response time of each request in the 2P algorithm is t_0, while the other three comparison algorithms are t_i, the reduction ratio is

$$\Upsilon = 1 - \frac{t_0}{t_i} \tag{12}$$

5.2 Results Comparing

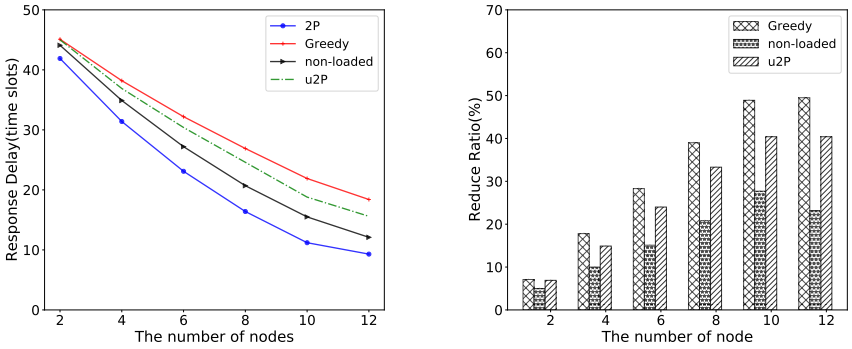


(a) Average response time per request (b) The reduction ratio of the 2P algorithm

Fig. 3. Change the node capacity R_n, where N=9 and θ=2

In Fig.3, we set the number of nodes as 9 and the upper limit of replicas as 2 in the system, but the amount of node resources increases gradually. According to Fig.3(a), we find that the delay time of all algorithms decreases with the increase of node resources. When the system node resources are sufficient, the response time remains stable. In addition, we also found that the greedy algorithm has the longest response time, the non-loaded algorithm has a smaller response time than the u2P algorithm, and the 2P algorithm has the shortest response time.

According to Fig.3(b), we get the response time reduction rate of the 2P algorithm compared with other algorithms. Within the ranges of [60, 90] and [90, 120], the response time of the 2P algorithm is about 46% lower than that of the greedy algorithm. Compared with the non-loaded algorithm, the response time was reduced by about 23%. Compared with the u2P algorithm, the response time is reduced by 37%. However, within the range of [1, 30] and [150, 180], the time reduction rate of the 2P algorithm is relatively low. This is mainly because resources are at two extremes (too scarce or too abundant).



(a) Average response time per request (b) The reduction ratio of the 2P algorithm

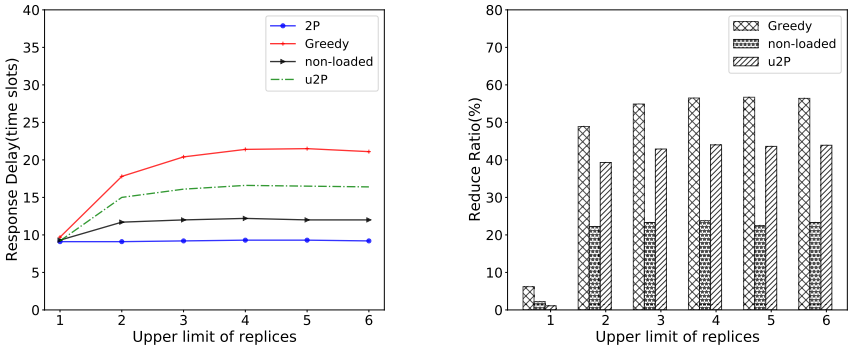
Fig. 4. Change the number of system nodes N, where θ=2 and $R_n \in$ [50, 100]

In Fig.4, we set the node resource to be within [50, 100] and the upper limit of replicas as 2 in the system. We infer that the 2P algorithm performs significantly better than the other three algorithms from 4(a). The response time of the non-loaded algorithm is lower than that of the u2P algorithm, indicating that the service load distribution variable has a greater impact on the response time. When the number of nodes increases gradually, the response time of all algorithms will decline significantly. The response time of the 2P algorithm is always the lowest.

In Fig.4(b), we konw that when there are two nodes in the system, the response time of the 2P algorithm decreases less than that of the comparison algorithm. This is mainly because the number of nodes is too small, resulting in an excessive shortage of system resources, which is consistent with the conclusion in Fig.3(b). However, with the increase of system resources, the reduction rate of the 2P algorithm increases gradually.

In Fig.5, we set the system to 12 nodes with each resource capacity in the [50, 100] range and change the upper limit of replicas. From Fig.5(a), the 2P algorithm has the shortest response time and the best performance based on stability. The greedy algorithm has the highest response time and the lowest performance. The delay time of the non-loaded algorithm and the u2P algorithm is larger than that of the 2P algorithm, but less than that of the greedy algorithm.

From Fig.5(b), we see that when the upper limit of the replica is 1, the reduction rate of the 2P algorithm is less than 10%. This is because the resource is very sufficient and is a limit state, which is consistent with the conclusion of Fig.3(b). When the upper limit of replicas is in the range of [2, 6], the response time reduction rate of the 2P algorithm is about 50% relative to the greedy algorithm. Compared with the non-loaded algorithm, the response time is reduced by about 23%. Compared with the u2P algorithm, the response time is reduced by about 24%.



(a) Average response time per request (b) The reduction ratio of the 2P algorithm

Fig. 5. Change the upper limit of service replicas θ, where N=12 and $R_n \in [50, 100]$

6 Conclusion

In this paper, we investigate the service placement for response delay reduction in a heterogeneous MEC system. We set priorities for the nodes and services, which can be adjusted dynamically according to the placed state. To model the priority, we analyzed several factors such as service load distribution or delay time, obtained the service placement strategy in order of priority. We conduct extensive simulations, and the results show that our 2P algorithm has significant performance improvement on response delay reduction.

Acknowledgment

This work is supported in part by the National Key R&D Program of China under Grant 2019YFB2102002, in part by the National Natural Science Foundation of China under Grant 61802182.

Reference

[1] R. Yu, V. T. Kilari, G. Xue, and D. Yang. Load balancing for interdependent iot microservices. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 298–306, 2019.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[3] Y. Li and S. Wang. An energy-aware edge server placement algorithm in mobile edge computing. In *2018 IEEE International Conference on Edge Computing (EDGE)*, pages 66–73, 2018.

[4] S. S. D. Ali, H. Ping Zhao, and H. Kim. Mobile edge computing: A promising paradigm for future communication systems. In *TENCON 2018 - 2018 IEEE Region 10 Conference*, pages 1183–1187, 2018.

[5] A. Reiter, B. Prünster, and T. Zefferer. Hybrid mobile edge computing: Unleashing the full potential of edge computing in mobile device use cases. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 935–944, 2017.

[6] Charith Perera, Yongrui Qin, Julio C. Estrella, Stephan Reiff-Marganiec, and Athanasios V. Vasilakos. Fog computing for sustainable smart cities: A survey. *ACM Comput. Surv.*, 50(3), June 2017.

[7] E. De Cristofaro and C. Soriente. Participatory privacy: Enabling privacy in participatory sensing. *IEEE Network*, 27(1):32–36, 2013.

[8] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck. Mobile edge computing potential in making cities smarter. *IEEE Communications Magazine*, 55(3):38–43, 2017.

[9] Jianjun Qiu, Xin Li, Xiaolin Qin, Haiyan Wang, and Yongbo Cheng. Utility-aware edge server deployment in mobile edge computing. pages 359–372, 01 2020.

[10] Kiryong Ha, Yoshihisa Abe, Zhuo Chen, Wenlu Hu, Brandon Amos, Padmanabhan Pillai, and Mahadev Satyanarayanan. Adaptive VM handoff across cloudlets. *Technical Report CMU-CS-15-113*, 2015.

[11] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas. Joint service placement and request routing in multi-cell mobile edge computing networks. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 10–18, 2019.

[12] S. Wang, M. Zafer, and K. K. Leung. Online placement of multi-component applications in edge computing environments. *IEEE Access*, 5:2514–2533, 2017.

[13] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser. Service entity placement for social virtual reality applications in edge computing. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 468–476, 2018.

[14] J. Xu, L. Chen, and P. Zhou. Joint service caching and task offloading for mobile edge computing in dense networks. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 207–215, 2018.

[15] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein. It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 365–375, 2018.

[16] S. Pasteris, S. Wang, M. Herbster, and T. He. Service placement with provable guarantees in heterogeneous edge computing systems. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 514–522, 2019.

[17] V. Farhadi, F. Mehmeti, T. He, T. L. Porta, H. Khamfroush, S. Wang, and K. S. Chan. Service placement and request scheduling for data-intensive applications in edge clouds. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1279–1287, 2019.

[18] Xin Li, Zhen Lian, Xiaolin Qin, and Jemal Abawajyz. Delay-aware resource allocation for data analysis in cloud-edge system. pages 816–823, 2018.