CCF Transactions on Pervasive Computing and Interaction (2021) 3:129–146
https://doi.org/10.1007/s42486-021-00066-2

REGULAR PAPER

Dual-label aware service replacement for interaction quality improvement in heterogeneous MEC system

Xin Li[1] · Meiyan Teng[1] · Jie Wu[2] · Xiaolin Qin[1]

Received: 1 December 2020 / Accepted: 1 April 2021 / Published online: 1 June 2021

Abstract
Interaction quality is an important factor for service provision to achieve better user experience. Mobile Edge Computing (MEC) is a promising paradigm to improve interaction quality by supporting near data computing at the edges. However, the limited resources at the edge nodes make it hard to response various services simultaneously, while the service load changes over time. Hence, it is important and challengeable to utilize the limited edge resources to host various service and reduce service response time to improve interaction quality. In this paper, we investigate the service replacement problem to adjust the edge resource utilization dynamically and then reduce the service response time. We first propose a priority placement (2P) algorithm to place the services among the edges by taking account the service priority, which indicates the influence for response time reduction. Then, we propose a dual-label aware service replacement (D-LASR) algorithm to achieve dynamic service placement to fit the service load variation. The replacement strategy works based on the delay sensitivity label and the load gradient label, which represent the features how the service location and service load affect the service response time. We conduct extensive simulations and the experimental results show that the D-LASR algorithm can reduce the average service response time by 40–60%, which indicates that the D-LASR algorithm has better performance in improving interaction quality for service provision in MEC system.

Keywords Heterogeneity · MEC · Service replacement · Interaction quality

1 Introduction

As the Internet of Things (IoT) technology matures and develops steadily, the Internet of Everything is rising. In the near future, the number of IoT application devices will reach tens of billions (Yu et al. 2019). Furthermore, delay-sensitive application devices that require real-time data processing, e.g., augmented reality (AR) and aided driving, are on the rise. The traditional cloud has been unable to meet the low latency response requirements of IoT applications (Shi et al. 2016).

Mobile edge computing (MEC), which offloads the cloud resources to the edge of the network, has drawn more and more attention (Li and Wang 2018). The MEC system reduces the response latency by reducing the distance between the requester and the server. The edge server can achieve real-time analysis, computing and send data that needs to be processed centrally to the cloud server. Edge computing is not completely replacing the cloud computing paradigm (Reiter et al. 2017), which merely adds another layer of computing between the users and the cloud. The edge layer exists not only reduces the burden on the cloud server, but also improves the user service experience.

It is predicted that there will be 54 million driverless cars in the world by 2035 (Perera et al. 2017). Cameras on vehicles for aided driving can generate approximately 1GB of data on road conditions per second (De Cristofaro and Soriente 2013), even the 5GB data will be produced by boeing 787 in a second (Taleb et al. 2017). For such data-intensive applications, the service requests not only impose strict demands on latency, but also consume a large amount of resources. Although the MEC system has relieved the bandwidth pressure in network transmission and reduced the response delay time of services, the resources (computing and storage, etc) capacity of edge service are limited

✉ Xin Li
lics@nuaa.edu.cn

[1] CCST, Nanjing University of Aeronautics and Astronautics, Nanjing, China

[2] Center forNetworked Computing, Temple University, Philadelphia, USA

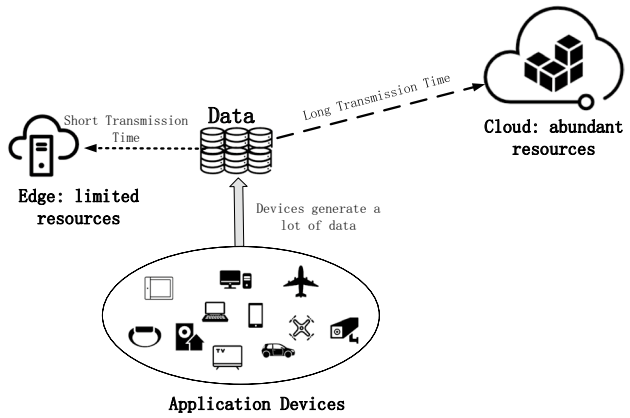


Fig. 1 Overview of cloud-edge in IoT environment

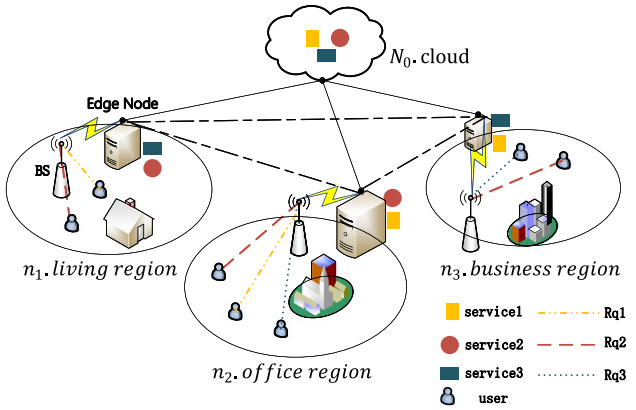


Fig. 2 System model

compared with those of the cloud. The insufficient resources of edge nodes will cause numberous applications services to be unresponded, thus reducing the user experience, as shown in Fig. 1. In order to make full use of edge node resources, more and more researches focus on service placement problem, which makes a trade-off between the cloud and the edge nodes to achieve the goal of minimizing response time.

In addition, diverse services have different requirements for response time. In life, services such as intelligent navigation and driverless cars have higher requirements for response time and need to provide accurate response results in a very short time. For intelligent navigation services, too long response time will reduce the quality of user experience. However, for driverless services, if the response time is too long, which will not only reduce the quality of user experience, but also cause personal safety issues for users. On the contrary, some services require relatively low response time, such as browsing the web and intelligent retrieval. For this type of service, when the edge node resources are limited, it can be placed in the cloud center to respond. Therefore, how to place services to ensure that highly sensitive services are placed on the local edge is also an urgent problem.

Due to the emergence of user mobility, the load of the service will be unevenly distributed over time. For example, the number of users in living and business regions is lower than that in office region on weekdays. Therefore, there are more requests for various kinds of services within the scope of the office region is more than those in the other regions. In other time periods, the distribution of users and the requests for services will adjust again. This example demonstrates that, in order to optimize the user experience of MEC, the resources required by the service for mobile users should be dynamically redistributed between edge nodes to adapt to changes in service load.

In this paper, we research a services replacement problem for interaction quality improvement in a heterogeneous MEC system, as shown in Fig. 2. In MEC system, the resources of cloud is sufficient to place all services, but the edge nodes has insufficient resources. The heterogeneity of MEC means that:

- the resource capacity of the edge nodes is difference;
- the services requirements in the system are discrepant;
- the network conditions are heterogeneous.

The MEC system is also interactive, which implies that:

- the requests can be responded on the cloud or the edge servers, e.g., the base station (BS) (Qiu 2019), etc.
- the edge nodes can communicate with each other, which will cause communication delay.

In this heterogeneous MEC interactive system, we studied the dynamic services replacement problem to improve the interaction quality. We divide the problem into two stages. First of all, we obtain the load of the service in the initial state through prediction, and study a static service placement strategy. Based on the static placement strategy of the initial state, the dynamic replacement strategy is studied in the subsequent stage according to the double label of the service load variation and the service sensitivity attribute.

In the initial state, we measure the value of nodes to a service placement problem based on the load distribution. The resources of nodes whose coverage contain many service requestes are limited, so that those nodes have higher value in improving the quality of service. When the request cannot be responded to at the local node due to node resource constraints, the request will be scheduled to another node to response. In response to this situation, we combine the load distribution and the network condition to difined the node priority. In addition, it is also a problem to determine the order of services placement. Therefore, we set service priorities on acccount of the heterogeneous attributes of services.

Then, we propose a priority placement (*2P*) algorithm to obtain a placement strategy with lower system latency.

In the subsequent state, we set the delay sensitivity label for the service according to the upper limit of the service response time. Then, we define a load gradient label for the service according to the variation of the load in the adjacent time period. Based on the dual label of delay sensitivity and load gradient, we determine the service resource redistribution strategy. Through experiments, we get the mapping relationship between load and response time under different resource conditions, which determines the granularity of service resource reallocation. We finally propose a service replacement algorithm based on dual labels.

The main contributions are summarized as follows:

1. We research the characteristics of the MEC model. And, in view of the heterogeneity and dynamics of system characteristics, we present the service replacement problem to improve the quality of service. We introduce the related work and prove the difficulty of this problem.
2. We propose a priority placement (*2P*) algorithm to achieve static services placement in the initial state. The core technology of *2P* algorithm is to measure the contribution of nodes and services in improving the quality of services based on the uneven load distribution, which determines the order of service placement.
3. We propose dual-label aware service replacement (*D-LASR*) algorithm to achieve dynamic service placement in the subsequent state. Based on the delay sensitivity label and the load gradient label, we classify services and implements different replacement strategies according to the category. We construct the mapping relationship between the load and the response time that determined the redistribution granularity.
4. We perform performance evaluations via simulation experiment. And we prove that the proposed algorithm has greater performance in proving the quality of services.

The remainder of this article is organized as follows. Section 2 overviews the related work on service replacement. Then, we introduce the system model and expound the service replacement problem in Sect. 3, and propose our service replacement strategy in Sect. 4. Next, we evaluate the performance of our algorithm in Sect. 5. Finally, we conclude the remarks in Sect. 6.

2 Related work

In recent years, the service replacement problem has attracted a lot of researches. More and more reseach results have emerged. In those researches, the most simplest method is that the services are placed on the local server, and the literature (Ha et al. 2015) demonstrates the effectiveness of this method. On the one hand, if the node resources are abundant, this method will minimize the response delay. On the other hand, if the resources are insufficient, most services will not be responsed on edge servers. In a comprehensive perspective, this method is not optimal.

The random rouding method is adopt in literature (Poularakis et al. 2019) to solve studies service placement problem under the scenario of overlapping node coverage. The authors in Wang et al. (2017) and Wang et al. (2018) neglect the limitations of the node resouce (computing resource, storage resource, and transmission bandwidth). Therefore, this strategy cannot achieve the best service response quality under the condition that the resource constraints. In our research, the coverage area of edge nodes is nonoverlapping and the resource of those is limited.

In MEC system, the node resources are not only limited, but also heterogeneous. But in literature (Xu et al. 2018), the authors prorose a service placement method through weighing the delay and cost under considering node isomorphism. The authors of paper (He et al. 2018) adopt greedy strategy and improve the system performance under the isomorphic environment. We research a MEC system including the difference of service requirements and the heterogeneity of node performance.

Literature (Pasteris et al. 2019) defines the reward based on the service quality, and divides node resources to select the service with the largest deployment reward. One disadvantage of this strategy is that multiple replicas of the same service are placed on the same node. And frequent placement operations consume a lot of power. The method of paper (Farhadi et al. 2019) is the linear programming, which jointly consider placement strategy and scheduling policy. But this research do not consider the condition of uneven service load distribution, which will cause a great impact on the service quality.

Due to the emergence of user mobility, service placement needs to be dynamically adjusted. For example, paper (Li et al. 2014) studied a migration decision according to the shortest path principle in Vehicle Ad-Hoc Networks. Similarly, the authors (Ksentini et al. 2014) use Markov Decision Processes to denote service migration problem and approximate the optimal solution based on Follow-Me Cloud and Mobile Micro-Clouds respectively. However, this study is merely limited on the single service and single user.

At present, most of the current work uses heuristic methods, such as the genetic algorithm (Islam et al. 2016), greedy method (Zhang et al. 2015), the regularization and rounding method (Lingjun et al. 2018). The work in Zhang et al. (2015) exploits the regularization technique to transform the optimization problem into a sequence of regularized sub-problems, then adopts the greedy strategy for request

redirection. And the authors in Hu et al. (2017) aim to balance the network delay and service load, solving the services placement problem in MEC.They formulated the service allocation problem and farther proposed a dynamic service allocation algorithm based on Pareto-based optimal k-Medoids to find an approximate optimal service allocation policy. Those works did not allocate resources according to different categories of service delay sensitivity attributes.

In our paper, we divide the service replacement problem into two stages. We reference previous work (Teng 2020) and define the priority of nodes and services based on the heterogeneous characteristics of nodes and services, then we propose a priority placement (*2P*) algorithm in the initial state. We also propose dual-label aware service replacement (*D-LASR*) algorithm to achieve dynamic service replacement in the subsequent states according to load variation and service sensitivity.

3 Problem statement

We concentrate on studying service dynamic replacement problem in a heterogeneous MEC architecture that is shown in Fig. 2. This system contains multiple edge server (node) and a centralized cloud. The cloud has sufficient storage resources to place abundant services and adequate computing resources that results in faster computing speed. However, the comumunication delay is too long due to the data source being farther from the cloud center and the network bandwidth limitation Therefore, the data-intensive and delay insensitive services are suitable to be placed on cloud. Furthermore, the edge nodes are close to data source. Therefore, if the services are responsed on edge nodes, it will produce lower delay time. However, the resources of edge nodes are limited. It is befitting to place delay sensitive services with low resource requirements on edge nodes.

The MEC system we studied is a heterogeneous interactive system. Services can communicate with each other between nodes or with remote clouds, but the network bandwidth conditions between nodes and the nodes resources are different. In this paper, the node set is signed by $N(n \in N)$, and the cloud is represented by *cloud*. The resource capacity of node is represented by the vector R_N, the distance between nodes is indicated by matrix $D_{N\times N}$, and the bandwidth condition is expressed by matrix $W_{N\times N}$. The matrices D and W are both symmetric. Each node $n \in N$ has a unique performance, as shown

- R_n means the resource capacity of node n;
- D_{nm} expresses the distance between the node n and the node m;
- W_{nm} represents the bandwidth condition between node n and node m;
- W_{c_n} represents the bandwidth condition between node n and *cloud*.

In addition, each service also has its own unique characteristics. The service set is $S(s \in S)$, and the data volume of the service is the vector da_s. Each service has different response time requirements. Services with high latency sensitivity need to be responded in a short time. Otherwise, the quality of service will be reduced. On the contrary, the long response time of the service with low delay sensitivity has less impact on its quality. In this paper, the vector sen_s represents the delay sensitivity of the service, and the vector ul_s represents the upper limit of the service response time. Moreover, the Cartesian product of service set and node set in the system is $M = N \times S(< n, s > \in M, n \in N, s \in S)$.

In our paper, the services response time $Tt_{\langle M,N\rangle}$ includes calculation time Ct_M and transmission delay $It_{\langle M,N\rangle}$, as shown in Eq. 1 The calculation time Ct_M is depends on the service type and the capacity of service load, which will be described in Sect. 3.1 The relationship among the transmission delay It_M, the amount of service data da_s, the node distance D and the bandwidth condition W is defined as shown in Eq. 2 Since the matrices D and W are both symmetric, for the same service s, there is $It_{<<s,n>,m>} = It_{<<s,m>,n>}$.

$$Tt_{\langle M,N\rangle} = Ct_M + It_{\langle M,N\rangle} \tag{1}$$

$$It_{\langle <s,n>,m\rangle} = \begin{cases} \frac{D_{nm}}{W_{nm}} \times da_s, & n \neq m \\ 0, & n = m \end{cases} \tag{2}$$

In a heterogeneous MEC environment, the load of services will change over time. Based on this feature, we study service dynamic replacement strategies. Firstly, we construct a time-slotted model (as shown in Fig. 3). The time model can be regarded as a simple sampling version of the traditional time model, and sampling is performed in the same time interval. The load distribution matrix in the time slot t is defined as $L_t(M)$. In a time slot, the load of the service fluctuates slightly, so we set the load $L_t(M)$ as the maximum value of the load in this slot, which remains fixed during same slot. However, the fluctuation range of the load from one slot to

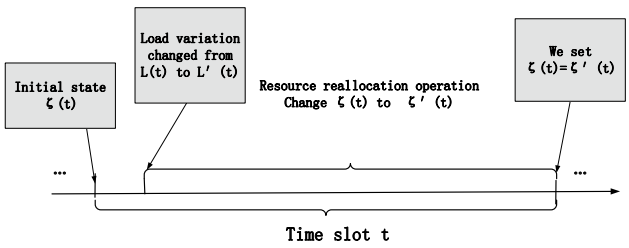


Fig. 3 Time slot

the next is uncertain. We use $\Delta L_t(M) = L_t(M) - L_{(t-1)}(M)$ to represent the load variation at time slot t. In the case of certain resource conditions, the increase in service load will reduce the response time of the service. In order to avoid a decrease in the quality of services in the system due to increased load, we need to dynamically adjust the amount of resources obtained by the service. Next, we get the mapping relationship between response time and load through experimental tests.

3.1 Service load profile

We place services on a real cluster server and use JMeter to simulate different numbers of requests to access the services at the same time. The experiment obtains the mapping relationship between service request response time and load under different resource conditions, as shown in Fig. 4. Figure 4a, b respectively count the load-time mapping of different types of services. We can receive from the figure that under the same resource conditions, as the load increases, the response time of service requests will also increase. When the load of the service is constant, the more resources allocated to the service, the lower the response time of the request. But when the amount of resources is infinite, it does not mean that the response time of the service request can be infinitely small. Because each service will have a response time lower limit dl_s, when the resource is saturated, the response time of the request tends to be stable and remains unchanged. In addition, from Fig. 4a, b, we also obtain that the load-time mapping is a monotonically increasing function. Therefore, suppose the mapping relationship between load and response time is:

$$\tau \rightarrow g(L) \tag{3}$$

In addition, in Fig. 4c, we compare the load-time mapping of different service types under the same resource conditions. From the Fig. 4c, we get that the load-time mapping is related to the service type. From the above experimental data, we can conclude that the calculation time of the service Ct_M is determined by the load L, service type and resource π_M. Therefore, we can abstract the mapping relationship among the four variables as follows:

$$Ct_M \rightarrow f\left(\text{sen}_s, g(L), \pi\right) = \frac{ksen_s \cdot g(\text{L})}{\pi_M}, \left(Ct_M > dl_s\right) \tag{4}$$

where, the parameter $ksen_s$ is determined by the service type. In addition, the service type is related to the service delay sensitivity sen_s and the variation of load. Next, we will define the double label of the service from the two factors of delay sensitivity and load variation.

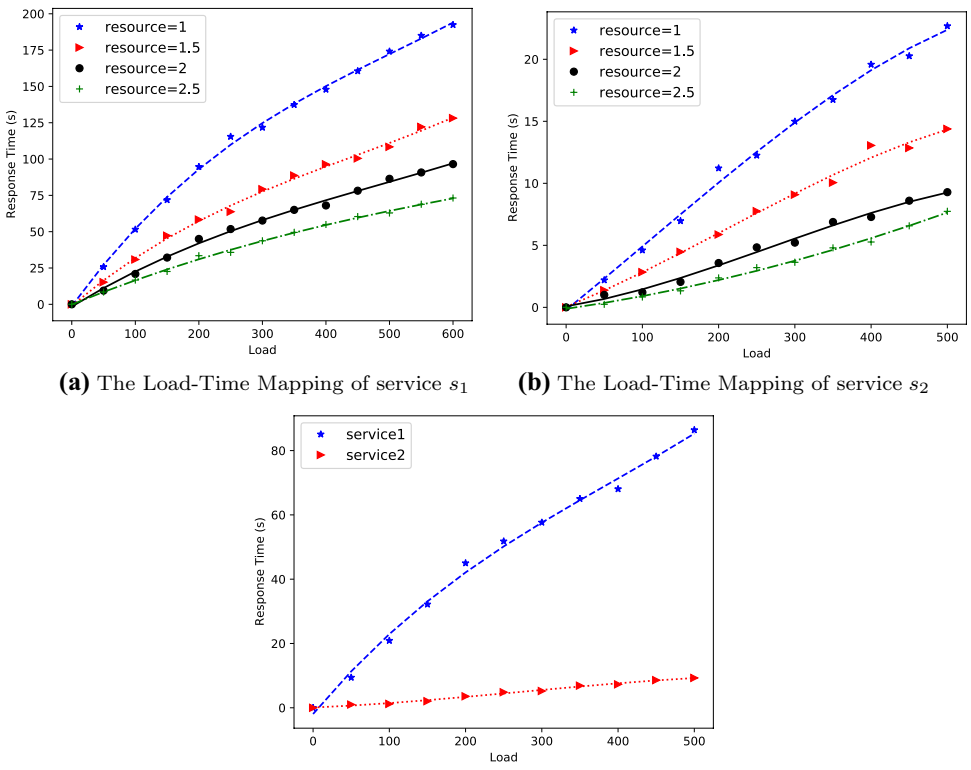


Fig. 4 The load-time mapping

(a) The Load-Time Mapping of service s_1

(b) The Load-Time Mapping of service s_2

(c) Set $resource = 2$

3.2 Service dual-label feature

The first label is delay sensitivity. According to the upper limit of service response time ul_s, we divide the system's services into two categories: high delay sensitivity and low delay sensitivity. Set a threshold value V_s for the response time of the service in the system, which is set to the maximum value of the transmission delay between the nodes and the cloud in the system, as in Eq. If the upper limit of the response time of the service is lower than the threshold value, add a high-sensitive label to the service and set $sen_s = 0$. If the upper limit of the service response time is not lower than the threshold value, add a low-sensitivity label to the service and set $sen_s = 1$. The symbol is defined as shown in Eq. 6.

$$V_s = \max_{n \in N} It_{\langle <s,n>,cloud \rangle} \tag{5}$$

$$s_{\text{sen}} = \begin{cases} 0, & ul_s < V_s \\ 1, & ul_s \geq V_s \end{cases} \tag{6}$$

In Fig. 4c, suppose service s_1 represents a low-sensitive service, and the upper limit of response time is ul_{s_1}. Service s_2 represents a high-sensitive service, and the upper limit of response time is ul_{s_2}, then $ul_{s_1} > ul_{s_2}$. In the case of limited resources and similar loads on s_1 and s_2, the high-sensitive service s_2 is allocated priority.

The second layer of label is the load gradient. We divide all services in the system into three categories: recycle, maintenance, and allocation based on the variation of the total load. The variation of the total load is measured by load change rate σ_s, which is defined as follows:

$$\sigma_s = \frac{\sum_{n=0}^{|N|} \Delta L_t(s,n)}{\sum_{n=0}^{|N|} L_t(s,n)} \tag{7}$$

The value of load gradient loa_s is $-1, 0, 1$, corresponding to three types of labels: recycle, maintenance, and allocation. This article defines the load gradient σ_s by setting a threshold, as follows:

$$loa_s = \begin{cases} -1, & \sigma_s < -k \\ 0, & |\sigma_s| \leq k \\ 1, & \sigma_s > k \end{cases} \tag{8}$$

The service delay sensitivity label sen_s and the load gradient label loa_s jointly determine the resource replacement method. Then the resource replacement granularity is jointly determined by the load amount L and the response time upper limit dl_s.

3.3 Problem formulation

As shown in Fig. 3, we define the state of the entire system at the beginning of each time slot as $\zeta(\mathrm{t}) = \left(\mathrm{L_t}, \mathrm{X_t}, \pi_\mathrm{t}, Y_\mathrm{t}\right)$. The state $\zeta(t)$ is named the initial state of the time slot t. Considering that the resource redistribution strategy π_t in the time slot t is determined based on the initial state $\zeta(t)$, node characteristics N, and service characteristics S. This paper uses $a_\pi(\zeta(t), N, S)$ to represent the reallocation of system resources when the system is in state $\zeta(t)$. At the beginning of each time slot, the MEC controller counts the change in service load from L_t to L'_t, $\Delta \mathrm{L_t} = \mathrm{L_t} - \mathrm{L}'_t$. When the load changes, the system's series of resource redistribution strategies are completed in the L'_t state. We assume that the time lost by the load change is negligible compared to the time lost by the resource redistribution (Fig. 3). Resource reallocation operation a_π causes the system state to change from $\zeta(t)$ to a new state $\zeta'(\mathrm{t}) = \left(\mathrm{L}'_t, \mathrm{X}'_t, \pi'_t, \mathrm{Y}'_t\right)$. The initial state of the next time slot $t+1$ is $\zeta(t+1) = \zeta'(\mathrm{t})$.

The matrix X_t represents the service placement strategy in the time slot t, the matrix π_t indicates the amount of resources allocated by the service, and the matrix Y_t expresses the response time of the service. Where, $x_t(s,n) = 1$ means that in the time slot t, the service s is placed at node n, the allocated resource is $\pi_t(s,n)$, and the response time is $y_t(s,n)$. On the contrary, $x_t(s,n) = 0$ means that the service s is not placed at node n, and the allocated resource amount is set to $\pi_t(s,n) = 0$. However, the request cannot be responded to locally, and the request needs to be dispatched to other nodes or remote clouds, the corresponding response time is $y_t(s,n)$. We stipulate that if the service s is placed at the local node n, it will be responded locally, and the response time is based on the load-time mapping to calculated. Otherwise, a remote response is required, and the request is dispatched to the corresponding response node $m(x_t(s,m) = 1)$. Therefore, the request response time $y_t(s,n)$ for service s at node n can be expressed as

$$\begin{aligned} y_t(s,n) = \Theta\left(x_t(s,n) == 1\right) Ct_{<s,n>} + \\ \Theta\left(x_t(s,n) == 0\right)\left(Ct_{<s,m>} + It_{\langle <s,n>,m \rangle}\right) \end{aligned} \tag{9}$$

where, if E is true, $\Theta(E) = 1$; otherwise, $\Theta(E) = 0$. $Ct_{<s,n>}$ means the average calculation time of service s at node n.

The variable $It_{\langle <s,n>,m \rangle}$ expresses the transmission delay that the request for service s is dispatched from node n to node m. In time slot t, the total response time $\Gamma_{a_{\pi_t}}(Y_t)$ of all services in the entire system is defined as follows:

$$\Gamma_{\mathrm{a}_{\pi_t}}\left(Y_t\right) = \sum_{n=0}^{|N|} \sum_{s=0}^{|S|} y_t(s,n) \tag{10}$$

Starting from the initial state $\zeta(0) = \zeta_0$, according to the resource allocation strategy π and the request scheduling

response time Y jointly determine the sum of the long-term average response time of the request, which is expressed as follows:

$$\mathbb{T}_{\Pi}(\zeta_0) = \lim_{T\to\infty}\left\{\sum_{t=0}^{T} \Gamma_{a_{\pi_t}}(Y_t)\middle|\zeta(0)=\zeta_0\right\} \tag{11}$$

In this article, our research aim is to design a resource reallocation strategy that meets the limited resource constraints and minimizes the total long-term response time of requests from the initial state.

$$\mathbb{T}^*_{\pi}(\zeta_0) = \min_{\pi}\mathbb{T}_{\pi}(\zeta_0), \quad \forall\zeta_0 \tag{12}$$

$$\sum_{s=0}^{|S|} ra_t(s,n) \le R_n, \quad \forall t, \forall n \in \mathrm{N} \tag{10.1}$$

where the constraint (10.1) means that the sum of resources occupied by all services that placed on one edge node should not exceed its capacity.

Based on the above problem statement, the service replacement problem is actually a limited resources allocation problem. We estimate that the service replacement problem is an NP-hard problem (Li et al. 2018). Next, we will prove the hardness of this problem.

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

Proof This theory will be proved by a instance, and its special scene is set up by following assumptions: (1) We set the service replacement operation to be completed in a time slot, (2) The system architecture is composed with one cloud and one edge node, and (3) one node is not allowed to place multiple replicates. Based on the above assumptions, the most ideal solution is to place services on edge nodes to minimize response time. But, the lackness of edge node resource results that we ought to select some services to place on the edge node, while other services are placed on cloud. The objective is to minimize the service response time. This problem is similar to the typical knapsack problem.

The typical knapsack problem can be formalized as follows. The items set is represent by $E = \{e_i, 0 \le i \le n\}$, and the weight and the value of item e_i is shown as w_i and v_i, respectively. The knapsack problem is to select a subset E_s such that the total weight does not exceed the capacity W of knapsack and the total value is maximized. Afterwards, in the system we assumed, the services set is $S = \{s_l, 0 \le l \le n\}$. For each service $s_l \in S$, we set the resource required by s_l be r_l, and the response delay of s_l be y_l. The object is to select a subset S_s of services such that the total resource occupied by services does not exceed the edge node resource capacity R and the response delay is minimized.

We assume that there exist a strategy selecting a subset E_s such that $\sum_{e_i\in E_s} w_i \le W$, and $\sum_{e_i\in E_s} v_i$ is maximized. Then, we can choose a service $s_l \in S$ with $r_l = w_i$ and $\frac{1}{y_l} = v_i$ corresponding to a item $e_i \in E_s$. So we obtain a subset S_s with the shortest total response delay time. In addition, if we select a subset S_s of S to minimize the total response time, we can get the subset E_s to maximize the value. Because the knapsack problem is NP-hard, we infer that the services placement problem in a time slot is NP-hard. It further proves that the services dynamic replacement problem in MEC system is also an NP-hard problem. □

4 Service replacement strategy

In the initial state of time slot $t = 0$, because the load of the service in the system is unknown, it is impossible to calculate the service distribution based on the mapping relationship $Ct_M \to f(\mathrm{sen}_s, g(L), \pi)$ and service characteristics. We use the prediction method to obtain the load L_0 in the time slot $t = 0$, the resource required by each service s is recorded as r_s, and the calculation delay of the service at each node.

In the time slot $t = 0$, we define the priority of nodes and services according to the predicted load L_0 and the total response time matrix $Tt_{\langle M,N\rangle}$. The detailed steps are in the Sect. 4.1. In the time slot $t > 0$, the load will continue to change over time, and resources will be dynamically reallocated for the service according to the service attributes and the gradient of load change, as shown in the Sect. 4.2.

4.1 Initial static placement

In order to roughly shorten the service response time, we formulate the node priority and service priority based on the load distribution and network condition. Afterwards, the initial service static placement strategy is determined under the defined of priority combined with greedy method. So we called our method priority placement (*2P*) algorithm. In 2P algorithm, we select the nodes in order of priority from high to low to make placement strategy firstly. Next, we select services with high-priority from the candidate service set of node to place in turn. The priority will be adjusted dynamically with the placed status changes in the procedure of initial pacement, make a dynamic priority definition. In the *2P* algorithm, the definition of node priority and service priority is a key step. Next, we will explain them in detail.

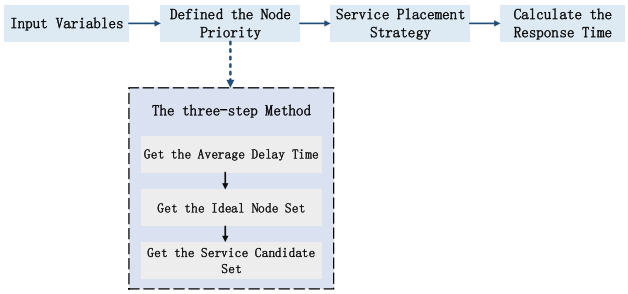


Fig. 5 Initial services placement process

4.1.1 Initial placement algorithm

The aim of research is to minimize the service response delay time, Fig. 5 describes the calculation process of the total delay time of system. Firstly, we define the input variables and set the performance parameter of nodes and services as well as the upper limit value of copies θ. In initial status, the service load distribution and the network candition are predicted through artificial intelligence technology. So, we use the symbol L_0 to represent the load distribution in time slot $t = 0$, and the symbol $Tt_{<M,N>}$ to express the system response time matrix. Then, we calculate the node priority through the three-step method, including geting the average delay time, geting the ideal node set and geting the service candidate set, which is introduced in detail in Sect. 4.1.2. Next, we get the initial placement strategy X_0 and variables of allocated resources π_0, which is introducted in detail in Algorithm 1.

Finally, we count the response time of all services in system at initial time slot $t = 0$, based on the initial placement strategy X_0. In time slot $t = 0$, the number of service replicas P_s is calculated according to the Eq. 13. The equation $P_s = 0$ means there has none replica of service s in system, and all request about service s should be scheduled to the cloud. For each node $n \in N$, we set $y_0(s,n) = Tt_{<<s,n>,cloud>}$. But, the equation $P_s \neq 0$ indicates that the system has one replicas of service s at least. In this case, the service request will be responsed on edge node whose response time is minimum. So, the response time $y_0(s,n)$ will be caculated by Eq. 14. Finally, output the placement scheme X_0, variables of allocated resources π_0 and response time Y_0.

$$P_s = \sum_{n=0}^{|N|} x_0(s,n) \tag{13}$$

$$y_0(s,n) = \begin{cases} Tt_{<<s,n>,cloud>}, & P_s = 0 \\ \min_{m \in N}\{Tt_{<<s,n>,m>}\}, & P_s \neq 0 \end{cases} \tag{14}$$

4.1.2 Define node priority

The number of services request distribute unevenly, which means that the service load distribution is imbalance. The resources of nodes whose coverage area includes a larger number of load are limited, and have greater contribution in improving the quality of services (QoS). The aim of our paper is to improve QoS through minimizing the total response time of all services, as expressed in Eq. 12. Therefore, our paper define node priority after three steps of calculation.

The first step, we suppose a situation that all requests for services s are responsed on one node m. Based on the input variables L_0 and $Tt_{<M,N>}$, we caculate the average response time when all requests of service s are scheduled to the same node m, which is formulated as follow Eq. 15. The average response time of other services can be counted in the same way. We take advantage of $averageDelay(Tt, L_0)$ function to calculate Eq. 15 and gain the matrix Q, which is a key variable in the *2P* algorithm.

$$Q_m^s = \frac{\sum_{n=0}^{|N|} Tt_{\langle\langle s,n\rangle,m\rangle} \cdot L_0(s,n)}{\sum_{n=0}^{|N|} L_0(s,n)}, s \in S, m \in N, n \in N \tag{15}$$

The second step is to caculate the ideal node set. Because the heterogeneity of system results in the $Q_m^s \neq Q_n^s (s \in S, m \in N, n \in N)$, the response time of request scheduling to different nodes is not equal. According to the order of matrix Q, the ideal node set is defined as follows:

$$\begin{aligned} NC_s = \Big\{ & n_{p_1}, n_{p_2} \cdots n_{p_i} \cdots | Q^s_{n_{p_1}} \leq Q^s_{n_{p_2}} \leq \cdots \\ & \leq Q^s_{n_{p_i}} \leq \cdots, n_{p_i} \in N \Big\} \end{aligned} \tag{16}$$

where $NC_{s,1} = n_{p_1}$ indicates that the first ideal node of service s is n_{p_1}, $NC_{s,2} = n_{p_2}$ indicates that the second ideal node of service s is n_{p_2}, …, and $NC_{s,i} = n_{p_i}$ indicates that the i^{th} ideal node of service s is n_{p_i}.

Get the ideal node set in Fig. 5 by calculatng a two-dimensional matrix $NC_{S\times|N|}$. In terms of the variables $NC_{s,i} \in NC$, where $s \in S, i \leq |N|$, we infer the meaning of row vector and column vector of two-dimensional matrix NC:

- row vector $NC_{s,|N|}$ indicates the ideal node sequence of service s;
- column vector $NC_{S,i}$ express the ith ideal node of all services.

Algorithm 1 Service Priority Placement (*2P*) Algorithm

Input: Node set **N**; Service set **S**; Average delay **Q**; Node candidate set **NC**; Service candidate set **SC**; Service load distribution $\boldsymbol{L_0}$; Variables of allocated resources $\boldsymbol{\pi_0}$; Upper limit of replicas θ.

Output: Service placement strategy $\boldsymbol{X_0}$, the variables of allocated resources $\boldsymbol{\pi_0}$.

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

The third step, based on the matrix *NC*, we get the initial service candidates set *SC* of each node, through *initServiceCandidateSet*() function. As follows:

$$SC_n = \left\{ s_{p_1}, s_{p_2} \cdots s_{p_i} \cdots \left| NC_{s_{p_i},1} = n, s_{p_i} \in S \right. \right\} \tag{17}$$

Initially, based on the vector NC_n, the service candidate vector of node *n* includes services whose ideal node is *n*, which is marked as SC_n. With the placement process, the reduction of node resource leads that the vector SC_n will be dynamically adjusted. The node whose service candidate vector SC_n has more services creates higher value on improving QoS. Therefore, the node priority we defined is the modulo $|SC_n|$.

4.1.3 Priority placement (*2P*) algorithm

The Algorithm 1 describes the service placement process in detail, which maps to the function *servicePlacement*() of Algorithm 1. The placement strategy of node *e* is to choose services *s* with higher priority from service candidate vector SC_e. After the service *s* is placed on node *e*, we add this service to the vector $SC_{e'}$ of its sub-ideal node e'.

The algorithm loops through lines 1–21 until the services candidate vector of all nodes is null. In a loop, we firstly choose a node with the highest priority in turn, which is expressed as *e*. Then, we sort the services within a collection SC_e by service priority Ω. The definition of service priority comprehensively considers three factors, the service loads, the number of replicas, the average response time and the response time gap with the sub-ideal node, as shown in the Eq. 18.

$$\Omega_{s,e} = \frac{\Delta Q_s + k_1 \cdot \sum_{n=0}^{|N|} L_0(s,n)}{Q_e^s + k_2 \cdot P_s}, s \epsilon S, e \epsilon N, n \epsilon N \tag{18}$$

In Eq. 18, the total load of the service *s* is caculated through $\sum_{n=0}^{|N|} L_0(s,n)$. The symbol P_s is the number of replicas of service *s*. Suppose that the ith ideal node of service *s* is node *e*, $NC_{(s,i)} = e$, and the next ideal node is e', $NC_{(s,i+1)} = e'$. So, the average response time indicated as symbol Q_e^s when all requests of services *s* are scheduled on node *e*. And the response time gapcaculate through $\Delta Q_s = Q_{e'}^s - Q_e^s$. The symbol k_1 and k_2 is parameter values. From the Eq. 18, we get a conclusion that the greater the time gap and the service loads are, the higher the priority is. And if the average delay time and the number of replicas are greater, the service priority will be lower.

Next, in lines 5–18, the algorithm select service based on the service priority to execute the placement process. If the resource requested by the service *s* is lower than

the remaining resources of the node e, the service will be placed and make $x_0(s,e) = 1$, $\pi_0(s,e) = r_s$. Otherwise, we set $x_0(s,e) = 0$ and $\pi_0(s,e) = 0$. In the next step, we judge whether to continue to place service s depends on the upper limit of service replicas θ. In lines 14-18, the service whose quantity in system is not more than the upper limit θ will be appended to the service candidate collection $SC_{e'}$ of the next ideal node e'. It is obtained by function *findNextNode*().

When the service is added to the service candidate collection of node e', the node priority $|SC_{e'}|$ and the service priority $\Omega_{s,e'}$ will change based on the Eqs. 17 and 18 respectively. At the end of the node e placement process, we clean up all services in the collection SC_e through the function *clear*(). And continue the placement process of next node in loops 1–21 until all nodes in system have none priority.

4.2 Dynamic service replacement

Under the constraints of the limited resources of the edge node, in order to allow more services to meet the upper limit of response time requirements, enhance the quality of service, so that achieve the purpose of minimizing system response time, we only place one low-sensitive service replica in the entire edge node system. All requests for the service within the scope of each node are dispatched to the same node to respond. Since the high-sensitive services need to be responded as quickly as possible, we try to place this type of service locally to avoid the degradation of service quality due to transmission delays. Next, we will introduce in detail how to use the dual-label aware service replacement algorithm based on service attributes and load gradient.

4.2.1 Dual-label aware service replacement (*D-LASR*) algorithm

Algorithm 2 describes the replacement strategy of resources in the continuous time slot, which is named dual-label aware service replacement algorithm (*D-LASR*). Initially, line 1 initializes the resource amount of the service at each node to zero ($\pi_t(M) = 0$). Line 2 divides the service set S into a highly sensitive service set *SH* and a low sensitive service set *SL* according to the delay sensitivity label sen_s of the service. Based on the load gradient label loa_s, line 3 divides the services in the SH and the *SL* into the high-sensitive recycle set *SHr*, the high-sensitive maintenance set *SHm*, the high-sensitive allocation set *SHa*, the low-sensitive recycle set *SLr*, the low-sensitive maintenance set *SLm* and the high-sensitive allocation set *SLa*. When reallocating node resources in each time slot, all resources of high-sensitive services that was allocation in the initial state must be recycle, such as lines 5–11. The purpose of this step is to avoid too many resources occupied by the service with too low load, and the service with overload cannot obtain enough resources, which will cause a waste of resources.

Algorithm 2 Dual-label Aware Service Replacement Algorithm (*D-LASR*)

Input: Node set $\mathbf{N}$; Service set $\mathbf{S}$; Service Load $\mathbf{L}$; The node distance metric $\mathbf{D}$; And the bandwidth condition $\mathbf{W}$.
Output: The total response time $\boldsymbol{\Gamma}$.

1: **for** $t = 1...T$ **do**
2: Initialize $\pi_t(M) = 0$ for all $M = N \times S$, $N = 0, 1, \ldots, N$; $S = 0, 1, \ldots, S$;
3: According to the sen_s, the S is classified into SH,SL;
4: According to loa_s, the set SH,SL are classified into SHr, SLr, SHm, SLm, SHa and SLa;
5: **for** each $s \in \mathbf{S}$ **do**
6: **for** each $n \in \mathbf{N}$ **do**
7: **if** $\pi_{t-1}(s,n) \neq 0$ **then**
8: $R_n = R_n + \pi_{t-1}(s,n)$;
9: **end if**
10: **end for**
11: **end for**
12: $RaLSS(SLr, \mathbf{X}, \mathbf{L}, \mathbf{Y}, Ct_M, \mathbf{S}, \mathbf{N})$;
13: $RaHSS(SHr, \mathbf{X}, \mathbf{L}, \mathbf{Y}, Ct_M, \mathbf{S}, \mathbf{N})$;
14: $RaHSS(SHm, \mathbf{X}, \mathbf{L}, \mathbf{Y}, Ct_M, \mathbf{S}, \mathbf{N})$;
15: $RaHSS(SHa, \mathbf{X}, \mathbf{L}, \mathbf{Y}, Ct_M, \mathbf{S}, \mathbf{N})$;
16: $RaLSS(SLm, \mathbf{X}, \mathbf{L}, \mathbf{Y}, Ct_M, \mathbf{S}, \mathbf{N})$;
17: $RaLSS(SLa, \mathbf{X}, \mathbf{L}, \mathbf{Y}, Ct_M, \mathbf{S}, \mathbf{N})$;
18: $\Gamma_{\mathrm{a}_{\pi_t}} = getTotalResponseTime(\mathbf{Y})$;
19: **end for**
20: $\Gamma = \sum_{t=o}^{T} \Gamma_{\mathrm{a}_{\pi_t}}$;

Next, lines 12–17 began to execute resource reallocation operations on services with various tags in turn. During this operation, the dual-label determines the sequence of resource reallocation operations. In this strategy, we give priority to allocating resources for the services in the low-sensitivity recycling collection, as shown in line 12. At first, the purpose of allocating services in the *SLr* set is to allow low-sensitivity services to release excessive resources and provide sufficient resources for subsequent services. In Algorithm 3, a detailed algorithm that reallocate low-sensitive service (RaLSS) is introduced. Following, lines 13–15 sequentially implement resource allocation for high-sensitive services. From the optimization goal, we stipulate that the priority order of resource allocation is $SHr > SHm > Sha$. In Algorithm 4, a detailed algorithm that reallocate high-sensitive service (*RaHSS*) is introduced. After that, lines 16–17 execute the *RaLSS* algorithm detailed in Algorithm 3 on the *SLm* and *SLa* sets in turn. Finally, line 18 uses the *getTotalResponseTime*() function to calculate the average response time $\Gamma_{a_{\pi_t}}$ of all requests in the t time slot, and line 20 calculates the average response time $\Gamma_{a_{\pi_t}}$ in all time slots in the continuous time T, which is used as an important indicator to measure the performance of our algorithm.

4.2.2 Reallocate low-sensitive services (RaLSS) algorithm

Algorithm 3 (*RaLSS*) realizes resource reallocation for low-sensitive services. In this algorithm, we use *SLx* to replace the three types of service sets *SLm*, *SLr*, and *SLa*. Due to the long response time required by low-sensitive services, there is only one replica of this kind of service in the system, and most of these service requests cannot be responded to locally. We need to schedule the request to the same node to respond. Although transmission delays occur during this process, for low-sensitive services, the final response time can meet the upper limit of service response time. In RaLSS algorithm, we first sort the services in *SLx* by priority in line 1. We use the total load to measure the priority of the service, such as the Eq. 19, the higher the total load, the higher the priority.

$$\mathrm{SP}_s = \sum_{n=0}^{|N|} L_t(s,n) \tag{19}$$

Next, lines 2–23 make resource reallocation decisions for the services in *SLx* in turn. The algorithm counts the total load *sumLoad* of the service s_i at line 3, and obtains the maximum value of the transmission delay *commTime* between each node in the system in line 4. Based on the sum variable of the load *sumLoad*, the difference between the upper limit of service response time s_{iul} and the maximum transmission delay commTime, and the mapping relationship $Ct_M \rightarrow f\left(\mathrm{sen}_s, g(L), \pi\right)$ between the calculated delay and service characteristics in the system, the *getMinResource*() function calculates the amount of resources on the line 5 of Algorithm 3. In line 6 of the algorithm, the *findProperNode*() function selects an optimal node from the system to place the service according to the amount of resources, load, and resource allocation at the previous moment $\pi_(t-1)$. In the function *findProperNode*(), we first find the node where the service s_i is placed from the matrix X_{t-1}. If it exists, select the node with the highest priority and the remaining resources $R_n \geq resource$. Otherwise, we select the node with the highest priority that satisfies $R_n \geq resource$ from the node set N in this system. We set that the higher the load rate and the more the remaining resources, the higher the priority the node has. The definition is as follows:

$$\mathrm{NP}_n = \frac{L_t(M)}{\sum_{n=0}^{|N|} L_t(s,n)} \times R_n \tag{20}$$

If a node n can be sought through *findProperNode*(), which can reallocate resources to the service s_i, execute lines 7–18, otherwise, execute line 19–22. If the node n allocates resource for the service s_i, it needs to update the related variables. For example, the amount of resources allocated to service s_i by node n is set to $\pi_t(s_i,n) = resource$, deployment variable is defined by $x_t(s_i,n) = 1$, and the remaining resources is set to $R_n = R_n - (\pi_t(s_i,n) - \pi_{t-1}(s_i,n))$, such as lines 8-10 in the algorithm. The variables π_t and x_t of other nodes are set to zero, such as lines 14-17. The response time $y_t(s_i,m)$ of the request for service s_i within the scope of each node is equal to the sum of the calculation delay time and the transmission delay $It_{<<s_i,m>,n>}$, which is calculated by *getResponseTime*() function. If the *findProperNode*() function does not find an appropriate node to allocate resources to the service s_i, all requests will be dispatched to the *cloud*.

Algorithm 3 Reallocate Low-Sensitive Services (RaLSS) Algorithm

Input: Node set $\mathbf{N}$; Service set $SLr(SLm, SLa)$; Service Load $\mathbf{L}$; Service placement strategy $\mathbf{X}$; Response time $\mathbf{Y}$; Load-Time Mapping Ct_M.
Output: The service placement strategy $\mathbf{X}$, The variables of allocated resources $\boldsymbol{\pi}$, The response time $\mathbf{Y}$.

1: order $SLx = s_1, s_2, \ldots, s_k$, so that $SP_{(s_1)} > SP_{(s_2)} > ... > SP_{(s_k)}$;
2: **for** each $s_i \in SLx$ **do**
3: $sumLoad = getSumLoad(\mathbf{L}, s_i)$;
4: $commTime = getMaxCommunicationTime(\mathbf{It}, s_i)$;
5: $resource = getMinResource(Ct_M, s_{iul} - commTime, sumLoad)$;
6: $n = findProperNode(\mathbf{X}, \mathbf{L}, \pi, \mathbf{N}, \mathbf{S}, resource)$;
7: **if** $n \neq -1$ **then**
8: $\pi_t(s_i, n) = resource$;
9: $x_t(s_i, n) = 1$;
10: $R_n = R_n - (\pi_t(s_i, n) - \pi_{t-1}(s_i, n))$;
11: $time = getResponseTime(Ct_M, resource, sumLoad)$;
12: **for** each $m \in \mathbf{N}$ **do**
13: $y_t(s_i, m) = time + It_{<<s_i,m>,n>}$;
14: **if** $n \neq m$ **then**
15: $\pi_t(s_i, n) = 0$;
16: $x_t(s_i, n) = 0$;
17: **end if**
18: **end for**
19: **else**
20: all requests for service s_i are responded to on the *cloud*;
21: for each $m \in \mathbf{N}$, set all $\pi_t(s_i, m) = 0$, $x_t(s_i, m) = 0$, $y_t(s_i, m) = time + It_{<<s_i,m>,cloud>}$;
22: **end if**
23: **end for**

4.2.3 Reallocate high-sensitive services (RaHSS) algorithm

For high-sensitive services, we place the service on a local server as much as possible to avoid transmission delay leading to a decline in service efficiency. Algorithm 4 (*RaHSS*) describes the complete strategy of resource allocation for high-sensitive services. Similarly, we use *SHx* instead of *SHr*, *SHm*, and *SHa*. First, we sort the services in the *SHx* set according to the priority of Eq. 19. Next, lines 2–20 sequentially redistributes resources within each node of each service in the *SHx* set. In the replacement process, if there is no request for service s_i in the range of node n, we will not do anythingthe. The three variables $\pi_t(s_i, n)$, $x_t(s_i, n)$ and $y_t(s_i, n)$ defaults to zero. Otherwise, we use the *getMinResource*() function to calculate the amount of resource required by the service, and use the *getResponseTime*() function to find the local response time in the state of serving this resource amount. Next, Algorithm 4 determines whether the remaining resource amount R_n of the local node n meets the resource required by the service. If it is satisfied, set the resource allocation amount $\pi_t(s_i, n) = resource$, placement variable $x_t(s_i, n) = 1$ and response time variable $y_t(s_i, n) = time$, and then update the remaining resource amount R_n of node n. If the remaining amount of the node does not satisfy the amount of resource required by the service, the requests are dispatched to the cloud.

Algorithm 4 Reallocate High-Sensitive Services (RaHSS) Algorithm

Input: Node set $\mathbf{N}$; Service set $SHr(SHm, SHa)$; Service Load $\mathbf{L}$; Service placement strategy $\mathbf{X}$; Response time $\mathbf{Y}$; Load-Time Mapping Ct_M.
Output: The service placement strategy $\mathbf{X}$, The variables of allocated resources $\boldsymbol{\pi}$, The response time $\mathbf{Y}$.

order $SHx = s_1, s_2, \ldots, s_k$, so that $SP_{(s_1)} > SP_{(s_2)} > ... > SP_{(s_k)}$;
for each $s_i \in SHx$ **do**
  for each $n \in \mathbf{N}$ **do**
    if $L_t(s_i, n) \neq 0$ **then**
      $resource = getMinResource(Ct_M, s_{iul}, L_t(s_i, n))$;
      $time = getResponseTime(Ct_M, resource, L_t(s_i, n))$;
      if $resource \leq R_n$ **then**
        $\pi_t(s_i, n) = resource$;
        $x_t(s_i, n) = 1$;
        $y_t(s_i, m) = time$;
        $R_n = R_n - \pi_t(s_i, n)$;
      else
        all requests for service s_i are responded to on the *cloud*;
        $\pi_t(s_i, n) = 0$;
        $x_t(s_i, n) = 0$;
        $y_t(s_i, n) = time + It_{<<s_i,n>,cloud>}$;
      end if
    else
      $\pi_t(s_i, n) = 0$;
      $x_t(s_i, n) = 0$;
      $y_t(s_i, n) = 0$;
    end if
  end for
end for

5 Evaluation

In this section, we evaluation *D-LASR* algorithm. We take the greedy algorithm as the baseline and conduct comparative experiments for three significant factors.

The first factor is the sensitivity of the service, so we constructed a comparison algorithm, namely the single-feature service reallocation algorithm, referred to as *S-FSR*. In the *S-FSR* algorithm, the services are classed into recycle set, maintenance set and allocation set. The detailed strategy of the *S-FSR* algorithm is to make redeployment decision for the services in the recycle set, maintenance set and allocation set in sequence. This algorithm stipulates that there is only one service replica in the system. Therefore, the *S-FSR* algorithm selects the optimal node to place for each service according to the node priority defined by Formula 20. Then it calculated the appropriate amount of resources for reallocation through the delay function Ct_M.

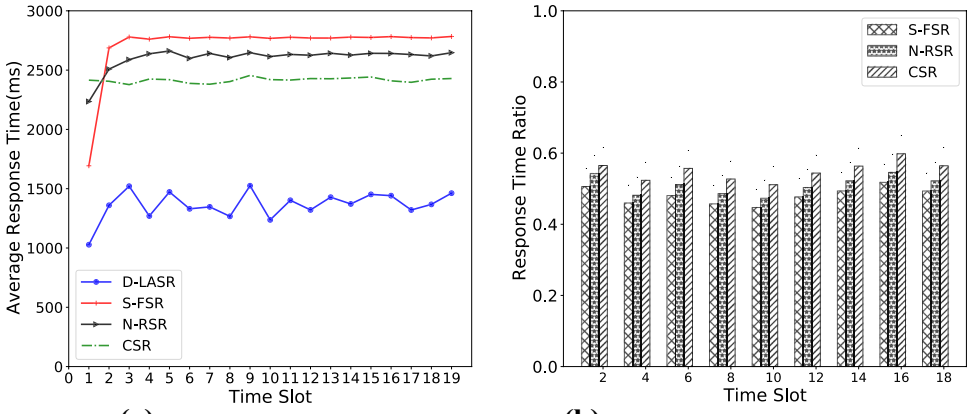
The second factor is whether to recycle useless resources, so we create a second comparison algorithm called the no-recycle service reallocation algorithm or *N-RSR* for short. The *N-RSR* algorithm reallocation strategy is to determine whether the amount of resources allocated at the previous time slot meets the amount of resources required at this time slot. If not, the service needs to be reallocation according to the delay function Ct_M.

In the third resource allocation algorithm, the amount of resources required by the service is constant and does not vary with changes in load, and each reallocation changes the node where the services are placed. We call this algorithm a constant service replacement algorithm, referred to as *CSR*. Next, 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. The resource of nodes R_n is set to be within the rage of [50, 350]. We set the transmission delay between nodes is $It_{<s,n>,m} \in [150, 250](n \neq m)$, which is a symmetric matrix. The resources in the cloud are infinite, and the transmission delay between the node and the cloud is set to $It_{<s,n>,m} \in [200, 400]$. We set up 100 services ($s = 100$) in the system, including two types of services, one is high latency sensitivity and the other is low latency sensitivity. For highly

Fig. 6 The response time



(a) The response time per time slot **(b)** The ratio of the response time

sensitive types, the upper limit of service response time is set to $ul_s \in [500, 600]$, lower limit of response time is set to $dl_s \in [100, 110]$. For lowly sensitive types, the upper limit of service response time is set to $ul_s \in [1000, 1100]$, lower limit of response time is set to $dl_s \in [200, 210]$.

This simulation experiment tests the reallocation of resources in 20 time slots. In each time slot, the load $L_t(M)$ is randomly generated in [0, 50], which obeys Gaussian distribution. The calculation time $Ct_t(M)$ is related to the service delay sensitivity, allocated resources and the amount of load, as shown in Eq. 4. For highly sensitive services, the parameter $ksen_s = 1$ in the formula, but for low sensitive services, the parameter $ksen_s = 40$. In our experiment, the mapping relationship between load and response time is set as $g(L_t(M)) = L_t(M)^{3/2}$.

The time slot $t = 0$ is the initial state, and the service placement strategy in this time slot is obtained according to the *2P* algorithm. In the initial state, because the service load is unknown, it is impossible to calculate the amount of resource that the service needed and the corresponding calculation delay according to the mapping relationship $Ct_M \rightarrow f\left(\text{sen}_s, g(L), \pi\right)$. In consequence, we set the resource demand of highly sensitive services $r_s \in [40, 50]$, and the resource demand of low sensitive services as $r_s \in [10, 20]$ by predicting the load distribution. In addition, the delay caused by computing is set to $Ct_0(M) \in [100, 200]$ at the time slot $t = 0$.

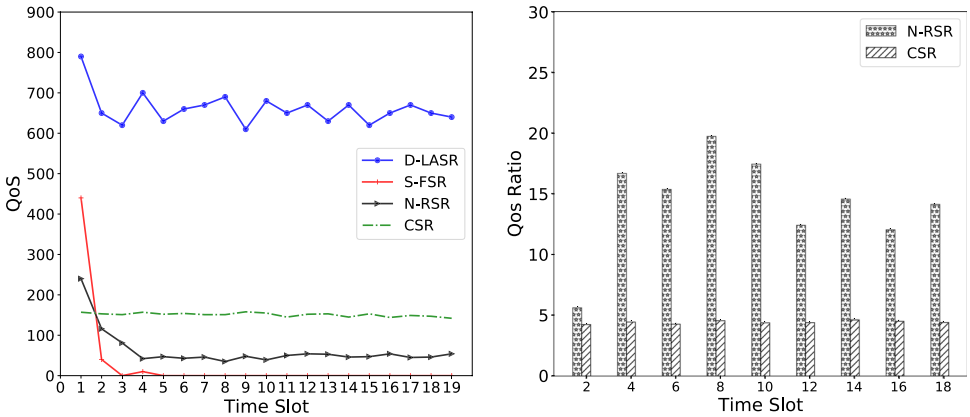
In order to better compare the performance of the *D-LASR* algorithm with the other three comparison algorithms, we conducted three series of comparison experiments. In addition to comparing response time variables Γ, we also compared the quality of service QoS and resource utilization σ_t.

In this article, the quality of service is measured by comparing the service response time $y_t(s, n)$ with the service response upper limit ul_s. If $y_t(s, n) \leq ul_s$, the quality of service is high, and set $QoS = 1$. Otherwise, set $QoS = 0$. The definition is as follows

$$QoS = \begin{cases} 1, & y_t(s, n) \leq ul_s \\ 0, & y_t(s, n) > ul_s \end{cases} \tag{21}$$

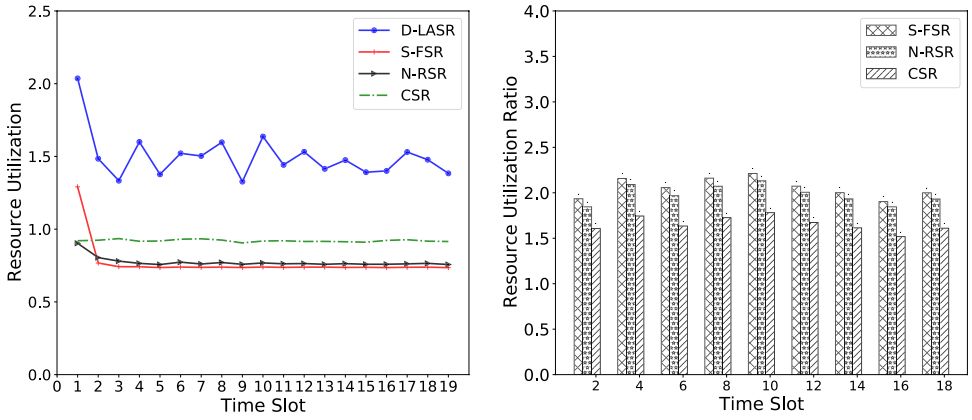
In addition, resource utilization is defined as follows

Fig. 7 The quality of services (QoS)



(a) The QoS per time slot **(b)** The ratio of the QoS

Fig. 8 The resource value



(a) The resource value per time slot **(b)** The ratio of the resource value

$$\sigma_t = \frac{\sum_{n=0}^{|N|} R_n + k}{\Gamma_{a_{\pi_t}}} \tag{22}$$

From the definition, we can see that if the remaining resources of the algorithm are the same, the lower the response time, the higher the resource utilization. Similarly, if the response time is equal, the more remaining resources, the higher the resource utilization. The k in the formula is a parameter, we set $k = 2000$ in our test.

At the same time, we calculate the ratio of the *D-LASR* algorithm and the three comparison algorithms in terms of response time, service quality, and resource utilization. Assuming that the performance of the *D-LASR* algorithm is P_0, while the other three comparison algorithms are P_i, the ratio is

$$\gamma = \frac{P_0}{P_i} \tag{23}$$

5.2 Results comparing

In Fig. 6, we count the average response time of each algorithm and the response time ratio between the *D-LASR* algorithm and the comparison algorithms in the time slot 1–19. According to Fig. 6a, we find that the average response time of services in the *D-LASR* algorithm is the lowest, fluctuating approximately between 1000 and 1500 ms. The response time of services in the *S-FSR* algorithm is the highest. This is because services are not placed according to delay sensitivity in the *S-FSR* algorithm. And fewer service replicas in this system cause that most requests to be dispatched to other nodes to respond, as well as, a large amount of transmission delay results in the increase of the total response time in system. In the *S-FSR* algorithm, the response time at the time slot $t = 1$ is smaller than that in the time slot $t > 1$. This is mainly because the remaining resource fragments of nodes in the system are less at the initial stage of resource reallocation, and most services can meet the demand, so the response time of the service is low. However, as time goes by, resource fragmentation in the system increases, causing many services to fail to respond at the edge, so the response time becomes longer and gradually stabilizes.

The response time of the *N-RSR* algorithm gradually increases in the time slot $t < 4$, but the response time tends to be stable in the time slot $t > 4$. The main reason for this phenomenon is that the *N-RSR* algorithm causes a lot of waste of useless resources, but the system resources are limited. As time goes by, more and more services will allocate resources that cannot reach the required amount of resources. When all resources in the system are occupied and none resources are allocated to the service, the response time of the system tends to be stable. Because the resource demanded by the service is stable in the *CSR* algorithm, the response time of this system is also constant. From the Fig. 6a, we discover that the average response time fluctuates around 2500 ms, which is higher than that of the *D-LASR* algorithm.

In Fig. 6b, we get the response time ratio between the *D-LASR* algorithm and the three comparing algorithms in the different time slots. The response time ratio is lower than zero, indicating that the response time of our algorithm is low and the performance is superior. From the Fig. 6b, we can find that the response time of the *D-LASR* algorithm is almost 1/2 of the comparison algorithm, and basically remains unchanged. In addition, since the response time ratio between the *D-LASR* algorithm and the *CSR* algorithm is the highest, however the response time ratio between the *D-LASR* algorithm and the *S-FSR* algorithm is the lowest, we can also get a conclusion that the *CSR* performance is the

best and the *S-FSR* algorithm has the worst performance in these three comparison algorithms.

In Fig. 7, we receive a curve graph of the quality of services (*QoS*) over time and a bar graph of the *QoS* ratio between the *D-LASR* algorithm and the comparing algorithms. According to Fig. 7a, we conclude that the *QoS* in the *D-LASR* algorithm is the highest, but in the *S-FSR* algorithm it is the worst and almost approaching zero. Corresponding to Fig. 6a, the *QoS* of the *S-FSR* algorithm is relatively high at the time slot $t = 1$. In the later stage, the nodes within the *S-FSR* algorithm generate a lot of resource fragmentation and the same service will be responded to on the same node, the remaining resources of the node cannot meet the service demand in this system. The above phenomenon causes that the actual response time of services exceed the upper limit, reducing the quality of service in the system. Similarly, the QoS in the *N-RSR* algorithm gradually decreases within the range of time slot $t < 4$, and tends to stabilize within the range of $t > 4$. The reasons for this trend are the same as in Fig. 6a. In addition, the *QoS* in the *CSR* algorithm is also changeless, and the conclusion of the experiment is consistent with that of the previous experiment.

Figure 7b shows the *QoS* ratio of *D-LASR* algorithm to *N-RSR* algorithm and *CSR* algorithm. The *QoS* ratio is greater than 1, indicating that the performance of the *D-LASR* algorithm is better than the *N-RSR* algorithm and the *CSR* algorithm. From Fig. 7a, we find that *QoS* of the *S-FSR* algorithm is approximately zero. Therefore, the *QoS* ratio between the *D-LASR* algorithm and the *S-FSR* algorithm is so large that we did not count it in the test of Fig. 7b. According to Fig. 7b, at the time slot $t = 2$, the *QoS* of the *N-RSR* algorithm is about five times as big as the *N-RSR* algorithm is. In the $t > 2$ stage, the *QoS* ratio fluctuates between 10 and 20 times. On the contrary, the *QoS* ratio between the *N-RSR* algorithm and the *CSR* algorithm is relatively stable and has been maintained at about four times. Based on the above conclusions, it can be concluded that the performance of the *CSR* algorithm is better than the *N-RSR* algorithm, which is consistent with the conclusion obtained in Fig. 6b.

In Fig. 8, we calculate the resource utilization rate of each algorithm over time according to the Eq. 22 and compare the performance of the *D-LASR* algorithm with the comparison algorithms in terms of resource utilization. It is found from Fig. 8a that the resource utilization rate of the *D-LASR* algorithm is the highest, and it remains around 1.5. The resource utilization rate of the *CSR* algorithm is lower than that of the *D-LASR* algorithm, and is about 0.9. In the time slot $t > 2$, the resource utilization of the *S-FSR* algorithm and the *N-RSR* algorithm are approximately equal. Although we find that the response time of the *N-RSR* algorithm is lower than that of the *S-FSR* algorithm from Fig. 6a, the resources of the *N-RSR* algorithm are more wasteful than the *S-FSR* algorithm. Therefore, according to the definition of resource utilization (Eq. 22), the similarity of resource utilization between *S-FSR* algorithm and *N-RSR* algorithm is reasonable.

Figure 8b shows the ratio of *D-LASR* algorithm and comparison algorithm in terms of resource utilization. The resource utilization ratio greater than 1 indicates that our algorithm has a higher resource utilization than the comparison algorithm. From this figure, we can see that the resource utilization of the *D-LASR* algorithm is approximately twice that of the *S-FSR* algorithm and the *N-RSR* algorithm. It can also show that the performance of the *S-FSR* algorithm and the *N-RSR* algorithm are similar in terms of resource utilization. In addition, the resource utilization of the *D-LASR* algorithm is about 1.6 times that of the *CSR* algorithm, which shows that the performance of *CSR* is better than that of the other two comparison algorithms in terms of resource utilization. Those conclusions are consistent with the previous two test conclusions.

6 Conclusion

In this paper, we investigate the service replacement problem in a heterogeneous MEC interactive system by dividing the problem into two states. In the initial state, 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 initial service placement strategy in order of priority. In the sequence states, we propose dual-label aware service replacement (*D-LASR*) algorithm to achieve dynamic service replacement that varies with service load. The delay sensitivity label and the load gradient label determine the replacement strategy, and the mapping relationship between the load and the response time determine the replacement granularity. We conduct extensive simulations, and the results show that our algorithm has significant performance improvement on response delay reduction, the quality of service (QoS) and the resource value.

Acknowledgements 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, and in part by the Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

De Cristofaro, E., Soriente, C.: Participatory privacy: enabling privacy in participatory sensing. IEEE Netw. **27**(1), 32–36 (2013)

Farhadi, V., Mehmeti, F., He, T., Porta, T.L., Khamfroush, H., Wang, S., Chan, K.S.: Service placement and request scheduling for data-intensive applications in edge clouds. In: IEEE INFOCOM

2019—IEEE Conference on Computer Communications, pp. 1279–1287 (2019)
Ha, K., Abe, Y., Chen, Z., Hu, W., Amos, B., Pillai, P., Satyanarayanan, M.: Adaptive VM handoff across cloudlets. Technical Report CMU-CS-15-113 (2015)
He, T., Khamfroush, H., Wang, S., La Porta, T., Stein, S.: 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), pp. 365–375 (2018)
Hu, B., Chen, J., Li, F.: A dynamic service allocation algorithm in mobile edge computing. In: 2017 International Conference on Information and Communication Technology Convergence (ICTC), pp. 104–109 (2017)
Islam, M., Razzaque, A., Islam, J.: A genetic algorithm for virtual machine migration in heterogeneous mobile cloud computing. In: 2016 International Conference on Networking Systems and Security (NSysS), IEEE, pp. 1–6 (2016)
Ksentini, A., Taleb, T., Chen, M.: A markov decision process-based service migration procedure for follow me cloud. In: 2014 IEEE International Conference on Communications (ICC), pp. 1350–1354 (2014)
Li, Y., Wang, S.: An energy-aware edge server placement algorithm in mobile edge computing. In: 2018 IEEE International Conference on Edge Computing (EDGE), pp. 66–73 (2018)
Li, X., Wu, J., Tang, S., Lu, S.: Let's stay together: towards traffic aware virtual machine placement in data centers. In: IEEE INFOCOM 2014—IEEE Conference on Computer Communications, pp. 1842–1850 (2014)
Li, X., Lian, Z., Qin, X., Abawajyz, J.: Delay-aware resource allocation for data analysis in cloud-edge system, pp. 816–823 (2018)
Lingjun, P., Lei Jiao, X., Chen, L.W., Xie, Q., Jingdong, X.: Online resource allocation, content placement and request routing for cost-efficient edge caching in cloud radio access networks. IEEE J. Select. Areas Commun. **36**(8), 1751–1767 (2018)
Pasteris, S., Wang, S., Herbster, M., He, T.: Service placement with provable guarantees in heterogeneous edge computing systems. In: IEEE INFOCOM 2019—IEEE Conference on Computer Communications, pp. 514–522 (2019)
Perera, C., Qin, Y., Estrella, J.C., Reiff-Marganiec, ., Vasilakos, A.V.: Fog computing for sustainable smart cities: a survey. ACM Comput. Surv., **50**(3), (2017)
Poularakis, K., Llorca, J., Tulino, A. M., Taylor, I., Tassiulas L.: Joint service placement and request routing in multi-cell mobile edge computing networks. In: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, pp. 10–18 (2019)
Qiu, J., Li, X., Qin, X., Wang, H., Cheng, Yo.: Utility-aware edge server deployment in mobile edge computing. Presented at the (2019)
Reiter, A., Prünster, B., Zefferer, T.: 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), pp. 935–944 (2017)
Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)
Taleb, T., Dutta, S., Ksentini, A., Iqbal, M., Flinck, H.: Mobile edge computing potential in making cities smarter. IEEE Commun. Mag. **55**(3), 38–43 (2017)
Teng, M., Li, X., Qin, X., Wu, J.: Priority based service placement strategy in heterogeneous mobile edge computing. Presented at the (2020)
Wang, S., Zafer, M., Leung, K.K.: Online placement of multi-component applications in edge computing environments. IEEE Access **5**, 2514–2533 (2017)
Wang, L., Jiao, L., He, T., Li, J., Mühlhäuser, M.: Service entity placement for social virtual reality applications in edge computing. In: IEEE INFOCOM 2018—IEEE Conference on Computer Communications, pp. 468–476 (2018)
Xu, J., Chen, L., Zhou, P.: Joint service caching and task offloading for mobile edge computing in dense networks. In: IEEE INFOCOM 2018—IEEE Conference on Computer Communications, pp. 207–215 (2018)
Yu, R., Kilari, V.T., Xue, G., Yang, D.: Load balancing for interdependent iot microservices. In: IEEE INFOCOM 2019—IEEE Conference on Computer Communications, pp. 298–306 (2019)
Zhang, X., Wu, C., Li, Z., Lau, F.C.M.: Online cost minimization for operating geo-distributed cloud cdns. In: 2015 IEEE 23rd International Symposium on Quality of Service (IWQoS), IEEE, pp. 21–30 (2015)

Xin Li received the B.S. and Ph.D degrees from Nanjing University in 2008 and 2014, respectively. Currently, he is an associate professor in the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics. His research interests include distributed computing, cloud computing and data management. He is a member of CCF.

Meiyan Teng received the B.S. degree from Nanjing University of Information Science & Technology in 2018, and the M.S. degree from Nanjing University of Aeronautics and Astronautics (NUAA) in 2021. Currently, she is a Ph.D. candidate at the College of Computer Science and Technology, NUAA. Her research interests include edge computing and edge intelligence.

Jie Wu is the director of the Center for Networked Computing and Laura H. Carnell professor at Temple University. He also serves as the director of International Affairs at College of Science and Technology. He served as chair of Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and associate vice provost for International Affairs from the fall of 2015 to the summer of 2017. Prior to joining Temple University, he was a program director at the

National Science Foundation and was a distinguished professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Mobile Computing, IEEE Transactions on Service Computing, Journal of Parallel and Distributed Computing, and Journal of Computer Science and Technology. Dr. Wu was general co-chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, ACM MobiHoc 2014, ICPP 2016, and IEEE CNS 2016, as well as program cochair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a fellow of the AAAS and a fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.

Xiaolin Qin is a professor in the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics. His research interests include data management and knowledge discovery.