Energy Efficient Service Composition with Delay Guarantee in a Cloud-Edge System

Quan Fang[1](✉), Menghan Xu[1], Hao Li[2], Jun Yu[3], Xin Li[2], and Zhuzhong Qian[4]

[1] Infomation and Telecommunication Branch, State Grid Jiangsu Electric Power Co., Ltd., Nanjing, China
[2] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China
{lh97,lics}@nuaa.edu.cn
[3] Nari Group Corporation, Nanjing, China
yujun@sgepri.sgcc.com.cn
[4] Department of Computer Science and Technology, Nanjing University, Nanjing, China
qzz@nju.edu.cn

Abstract. In a cloud-edge system, mobile users submit comprehensive service requests, on-the-fly service composition to orchestrate service components from different edge nodes is a promising way to achieve a quick response to these requests. Since several mobile applications consume large amount of energy during waiting for the responses, it is critical to achieve less service delay for energy saving as well as improve QoE (Quality of Experience). However, the service completion time in an edge is quite unstable, which increases the overall response time of the composite service. This paper argues that we may accelerate services through service clone via different edges, so that we can guarantee the overall response time of the composite service. And since the data fetch is also time consuming, we propose an effective data-aware service composition algorithm via service cloning to minimize the overall response time. We implement the algorithm and evaluate the performance with extensive simulations. The simulation results show that the proposed algorithm has a good performance improvement on service delay and energy consumption reduction, compared to the traditional algorithms.

Keywords: Cloud-edge system · Edge computing · Energy efficient · Service composition · QoS

1 Introduction

With the rapid development of the Internet of Things (IoT) and wireless communication technology, the intelligent era of the Internet of Everything is accelerating, with which the data volume grows quickly [1]. This situation poses a huge challenge to the currently widely used cloud computing models [2–4]. Although the traditional cloud data center can provide powerful data analysis capability, the large volume of data transmission from mobile users to far away data center results in long service delay. In the meantime, mobile devices are energy-aware, and long service delay actually significantly increases the energy consumption, since most applications consume energy during waiting for the response [5]. Deploying services on the edges nearby is a promising way to reduce the service delay, and it also reduces the waiting time of applications on mobile devices. Thus, edge computing actually reduces the energy consumption via shorten the service delay.

In a cloud-edge system, many edges are resource-limited and only suitable for running certain service components. To achieve a comprehensive requirement, several service components need to be composed into a composite service. When a user's request is submitted to an edge via a mobile device, the edge will be in charge of this request and generate a composite service based on the predefined business logic to achieve the composite service [6]. In the running time, the edge orchestrates the composite service and coordinates involved edges to correctly proceed the data and return the final results to mobile users.

Since the resource limitation, edges are usually unstable. When the workload grows, the response time of edge increases, or even could not get response [7]. Thus, cloning service requests to more than one edge and could get response with the earliest one, through which we may guarantee the service response time in an uncertain environment. On the other hand, several services are data-intensive, thus, the service response time is also closely related to data transmission delay. The data transmission includes both data transfer among service components and the other required data that needed during the processing. How to choose suitable service components to construct an effective service path on-the-fly is a non-trivial resource scheduling problem.

With the rise of cloud-edge system, researchers have proposed a variety of service selection methods to support on-demand service composition. However, they assume that the network resources are over-provisioned and do not consider the use of these resources when making quality-aware service composition decisions [8,9]. In [10], the authors proposed a collaborative filtering-based service recommendation method applied in the mobile edge computing environment and performing QoS prediction based on user mobility. This method first calculates the similarity of users or edge servers and selects the k-nearest neighbours to predict service QoS, and then performs service recommendation. Reference [11] studied the task allocation problem of reducing service delay in cloud edge systems. The authors used W-DAG (Weighted Directed Acyclic Graph) to model data-intensive services or business logic, and analyzed the tasks that constitute integrated services.

In this paper, we investigate on-line service composition problem in a cloud-edge system. We try to minimize the overall composite service response time to reduce the energy consumption of mobile devices. Our main idea is to clone some key service components and construct more than one service path to guarantee the response time, with limited clone budget. And since the data fetch is also time and resource consuming, we also design a data-aware service selection strategy to further accelerate the composite service.

The main contributions are as follows:

- Model and formalize the service composition problem in a cloud-edge system.
- An energy efficient data-aware service composition algorithm is proposed to minimize the overall response time via service components cloning.
- Extensive simulations show that the proposed service composition algorithm can achieve a good performance in reducing response time, compared with the stochastic algorithm and the greedy algorithm.

This paper is organized as follows. Section 2 presents the system model and formalizes the service composition problem and Sect. 3 is the energy efficient service composition algorithm. We evaluate the performance of the algorithm in Sect. 4. Finally, Sect. 5 concludes this paper.

2 Problem Statement

For a typical cloud-edge system, it consists of terminal layer, edge computing layer and cloud computing layer [12–14], as shown in Fig. 1. There always be one cloud data center (DC) and multiple edge nodes (EN) in the cloud-edge system. Also, it is feasible to perform inter-layer and cross-layer communication through the network for each layer.

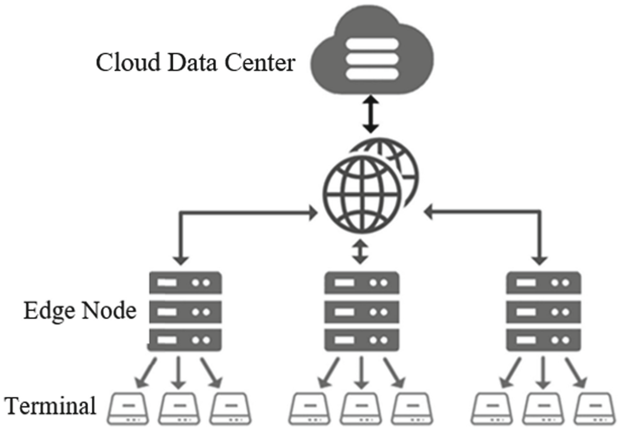

Fig. 1. Cloud-edge system.

To accelerate the service response and reduce energy consumption, some service component should be deployed on the edge, since the edge is close to the mobile users who submit the service requests. However, the edge nodes have limited resource and many service components must be placed on cloud data center due to the security issues.

To achieve a comprehensive service request, it is necessary to select some service components to construct a composite service. The classical method is to select some of the best service components with minimized service delay. However, it is challenging to construct the service path due to the uncertainty of edges, which may get unexpected long service delay.

In this paper, we aim to accelerate the service response by selecting more service components, which can generate more service paths. Thus, we can get the quick service response with high probability since any service path could response the service request. But select more service components means more computing and communication resources, hence, we need to investigate how to select the service components in a cloud-edge system with quantitative limitation due to budget constraint.

2.1 Service Path

We use the service path to represent the construction of a composite service, represented by F_i. An example of a service path is shown in Fig. 2. This composite service requires 4 service components. For each service component, several duplicated instances may be deployed in different edges in a cloud-edge system. Actually, what we need to do is to select a set of service component instances to achieve the service response. In the Fig. 2, the number above the service component indicates the number instance of the service component.

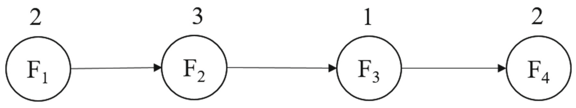

Fig. 2. An example of a service path.

2.2 Service Instance

The service instance is the service entity of each service component, represented by $s = <pos, ra, o, e>$, which is described by 4 attributes: pos represents the location of the service instance, and there are two options (DC/EN); ra represents the ratio of the amount of input and output data of the service instance, that is, $ra = amount_{output}/amount_{intput}$; o represents the time that the service instance spends processing unit data; e represents the effectiveness of the service (0/1), whether it is available or not. The relationship between the settings of o and e and the location of this instance is shown in Table 1. $mEN(s)$ in the table indicates the maximum number of service instances that can be selected on the node where service instance s is located. $pEN(s)$ represents the number of selected services currently owned by the node where the service instance s is located.

Table 1. Setting of o and e

	DC	EN
o	C_0	$N_o\left(\mu,\sigma_o^2\right)$
e	1	If $pEN(pos)+1 \leq mEN(pos) e=1$
e	1	If $pEN(pos)+>mEN(pos) e=0$

2.3 Service Graph

The service (instance) is labeled as $SG=<S,E,s_0>$. s_0 is the location of the initial data required to complete the composite service. When the composite service is requested, a corresponding SG will be generated according to the service path of the composite service. S represents the set of all service instances of the service components involved in the path; E represents the set of directed edges between service instances in S. Note that s_0 is added to S as a starting point.

2.4 Service (Instance) Chain ω

ω represents a service instance chain of the service path. For example, for a service path, $s_0 \rightarrow s_1^x \rightarrow s_2^x \rightarrow s_3^x \rightarrow s_4^x (1 \leq x \leq N(F_i))$ is an ω. The total number of service chains of a service path $path$ is denoted as $L(path)$, and the calculation method is shown in Eq. 1.

$$L(path)=\prod_{F_i \in path} N(F_i) \tag{1}$$

3 Service Composition Algorithm

The problem is to select some instances to form a composite service in the cloud-edge system. The most noteworthy problem is that the edges (servers) have limited resources, and the performance is unstable, which is different from the cloud data center with stable performance. Meanwhile, the network status between edge nodes and cloud data center is unstable. If only one instance chain is selected for a composite service, we can not guarantee the response time. If an edge is unable to respond due to limited resources or network communication terminals, etc., it will increase both service delay and energy consumption. Because service execution is very energy-intensive, for this problem, the first thing is to ensure that multiple chains are selected to complete a certain composition of tasks to prevent the above situation. For the selected multiple chains, their structures may have intersections (there are common service instances) or completely independent from each other (there are no common service instances), and they can be executed simultaneously at the same time.

3.1 The Delay of a Service Chain

In this paper, the delay of the instance chain set is used as the indicator to evaluate different service composition algorithms, and define the delay of the instance chain in the set that completes the composite task most quickly as the delay of the instance chain set. Therefore, we need to consider a calculation method of instance chain delay as shown in Fig. 3 (blue circle indicates that the service instance is deployed in the cloud data center).

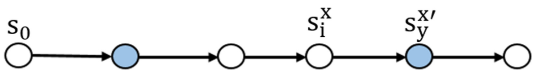

Fig. 3. Instance chain ω. (Color figure online)

D(ω) represents an ω delay, which consists of processing data time and transmitting data time. In order to calculate the delay of the whole chain, we first analyze the delay between two adjacent instances in the chain, as shown in Fig. 4.

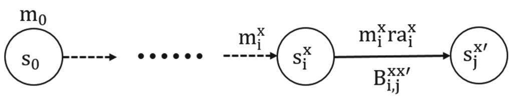

Fig. 4. Schematic diagram of data transmission and processing.

In Fig. 4, m_0 represents the initial amount of data at s_0, x represents the sequence number of the corresponding instance in its service class, $1 \leqslant x \leqslant N(F_i)$. m_i^x is the amount of data getting into the current service s_i^x, $m_i^x ra_i^x$ indicates the amount of data to be transferred to the subsequent service $m_j^{x'}$ after the current instance s_i^x has processed the data. $B_{i,j}^{xx'}$ indicates the bandwidth of data transmission. The setting of $B_{i,j}^{xx'}$ is related to the location of s_i^x and $s_j^{x'}$.

When both s_i^x and $s_j^{x'}$ are in the cloud data center or the same edge node, set $B_{i,j}^{xx'}$ to a larger value. If both s_i^x and $s_j^{x'}$ are in two different edges, or one is in the cloud center, the other is in the edge, then $B_{i,j}^{xx'}$ is regarded as a random variable, let it follow the Gaussian distribution of the mean and variance as the corresponding values.

Therefore, the part of processing data in the delay between s_i^x and $s_j^{x'}$ (recorded as $D_{i,j}$) is described by attribute o in Table. 1, and the part of transmitting data is described by $B_{i,j}^{xx'}$. Therefore, the calculation method of $D_{i,j}$ is shown in Eq. 2.

$$\begin{aligned} D_{i,j} &= \frac{m_i^x ra_i^x}{B_{i,j}^{xx'}} + m_i^x ra_i^x * o_j^{x'} \\ &= m_i^x ra_i^x \left(\frac{1}{B_{i,j}^{xx'}} + o_j^{x'} \right) \end{aligned} \tag{2}$$

In Eq. 2, $o_j^{x'}$ represents the time taken by $s_j^{x'}$ to process unit data. $D_{i,j}$ is actually the sum of the time of data transmission between s_i^x and $s_j^{x'}$ and the time of data processing in $s_j^{x'}$. At the same time, the $m_i^x ra_i^x$ in Eq. 2 can be determined by the initial data amount m_0, and the ra attribute of all service instances passing from s_0 to s_i^x on the current ω, this paper assumes that all service instances ra of an abstract service are the same, but their service processing time and network environment are different, so another expression of $m_i^x ra_i^x$ is shown in Eq. 3.

$$m_i^x ra_i^x = m_0 * \left(\prod_{s_j^x \in \omega_{s_0, s_i^x}} ra\left(s_j^x\right) \right) \tag{3}$$

In Eq. 3, $ra\left(s_j^x\right)$ represents the ra attribute value of s_i^x. Therefore, $D_{i,j}$ can be expressed in the form of formula 4.

$$D_{i,j} = m_0 \left(\prod_{s_j^x \in \omega_{s_0, s_i^X}} ra\left(s_j^x\right) \right) * \left(\frac{1}{B_{i,j}^{xx\prime}} + o_j^{x'} \right) \tag{4}$$

For Eq. 4, let $\eta_{i\omega} = \prod_{s_j^x \in \omega_{s_0, s_i^x}} ra\left(s_j^x\right)$, according to the previous settings, for a certain s_i^x on a given chain ω, $\eta_{i\omega}$ is a fixed constant. Set $\omega_{i,j} = \frac{1}{B_{i,j}^{xx\prime}} + o_j^{x'}$, note that except for the case that s_i^x and $s_j^{x'}$ are in the center of the cloud, other cases $\omega_{i,j}$ are random variables subject to a certain distribution. And $\omega_{i,j}$ represents the comprehensive performance of the network environment where a service is located and the data processing capability of the service. The smaller the $\omega_{i,j}$, the better. Thus, the form of $D_{i,j}$ can be reduced to Eq. 5.

$$D_{i,j} = m_0 \eta_{i\omega} * \omega_{i,j} \tag{5}$$

It should also be noted that s_0 only represents the location of the initial data, that is, only the data is transmitted and not processed. Based on the above, the total delay $D(\omega)$ of a service instance chain ω can be described in the form shown in Eq. 6.

$$\begin{aligned} D(\omega) = \sum_{e_{ij} \in \omega} D_{i,j} &= \sum_{e_{ij} \in \omega} m_0 * \left(\eta_{i\omega} * \omega_{i,j}\right) \\ &= m_0 \sum_{e_{ij} \in \omega} \left(\eta_{i\omega} * \omega_{i,j}\right) \end{aligned} \tag{6}$$

According to Eq. 6, it is also mentioned above that for each service instance of a specific chain, $\eta_{i\omega}$ is a constant value, and m_0 and o_0 are known constants. Therefore, in the intuitive form, to make the total delay of the selected instance chain as small as possible, it is necessary to make $\omega_{i,j}$ as small as possible.

3.2 Data-Aware Service Composition Algorithm

```
Algorithm 1: locAwareSelect(chainNum, preNode, Ii, pEN)
Input: the number of the current chain $chainNum$ , the previous instance
       $preNode$ , the candidate instance set $I_i$, the array pEN []
Output: Feasible service instance number
1  $I_i \leftarrow rank(l_i)$;
2  prePos = preNode.getPos();
3  samePosNum = findSamePos($I_i$);
4  if (samePosNum != -1) then
5      if ($I_i$.get(samePosNum).getPos() == 0 ||
        $I_i$.get(samePosNum).getRunFlag() == 1) then
6          return samePosNum;
7      else
8          while (pEN[$I_i$.get(samePosNum).getPos()] + 1 > MAX_INS) do
9              samePosNum = random.nextInt($I_i$.size() * 50%);
10             if ($I_i$.get(samePosNum).getPos() == 0 ||
                $I_i$.get(samePosNum).getRunFlag() == 1) then
11                 break;
12             end
13         end
14         return samePosNum;
15     end
16 else
17     choice = random.nextInt($I_i$.size() * 50%);
18     if ($I_i$.get(choice).getPos() == 0 || $I_i$.get(choice).getRunFlag() == 1) then
19         return choice;
20     else
21         while (pEN[$I_i$.get(choice).getPos()] + 1 > MAX_INS) do
22             choice = random.nextInt($I_i$.size() * 50%);
23             if ($I_i$.get(choice).getPos() == 0 || $I_i$.get(choice).getRunFlag() ==
                1) then
24                 break;
25             end
26         end
27         return choice;
28     end
29 end
```

We propose an energy efficient data-aware service composition algorithm. In fact, from the perspective of the algorithm execution process, there are also greedy algorithms and random ideas: when you want to select the next service instance, first check whether there is one in the same place as the previous instance (both in DC or the same EN) from the candidate set. If there is one, it is preferred; if there is no such instance in the current candidate set, or because of some limiting factors, there is the same location as the previous instance, but the instance cannot be selected, the greedy algorithm is used to select the better one from the remaining instances. The pseudo code is shown in Algorithm 1.

$Rank(I_i)$ in the first line of Algorithm 1 means that the current candidate instance set is sorted from small to large according to its $\omega_{i,j}$; *prePos* is the location of the previous instance; $findSamePos(I_i)$ is to find the same instance sequence number in the candidate set as the *preNode* position, and return the instance number if there is one, or -1 if it is not; line 9 means that the current candidate set has the same instance as the *preNode* position, However, due to conditional constraints, it is not possible to select an instance. Then randomly select the first 50% of the set of instances in the instance set that have already sorted the $\omega_{i,j}$ according to the number of candidate instances, which combines the stochastic algorithm and the greedy algorithm.

4 Evaluation

4.1 Simulation Settings

In the simulation environment, we set up one cloud data center (DC), and 5 edges (1–5). Therefore, the value of *pos* for all service instances is 0∼5, where 0 represents the cloud data center; and $1 \sim 5$ represents the corresponding edge nodes. Service path $path$:$F_1 \rightarrow F_2 \rightarrow F_3 \rightarrow F_4$, and its $N(F_i)$ is $3, 4, 2, 5, 3$, which means that the number of service instances is $3, 4, 2, 5, 3$.

Besides, the parameters that need to be set for each instance in this experiment are: location (*pos*), ratio of output to input data (*ra*), unit data processing time (*o*); meanwhile, according to the previous delay analysis, in order to simplify the programming model, the network environment of the instance (that is, the bandwidth allocated to an instance) is also incorporated into the parameters of the instance. Since the edges are closer to users, while the computing capacity of a cloud is better, the general principle of parameter setting is that when the service instance is deployed at the edge: the unit data processing time is longer, the network bandwidth may be better; when the service instance is deployed at the cloud center: the unit data processing time is short, the network bandwidth may be poor.

4.2 Results Analysis

This section mainly introduces comparative experimental methods, processes, and corresponding results display and analysis instructions. The purpose of the experiment is to compare the performance of three service composition algorithms including a stochastic algorithm, a greedy algorithm, and a data-aware service composition algorithm under the same constraints and experimental scenario settings. We investigate the performance of the algorithms by the delay of service instance chain, which actually indicates the energy consumption of the mobile devices. The comparison method is to simulate the running of the instance chain set selected by different methods more than 10,000 times, and the parameters are randomly generated according to the respective obedience distribution, and the delay is plotted into a CDF (distribution function diagram). Generally, the curve, that is closer to the y-axis, means less service delay and less energy consumption.

The main factors that affect the delay of the instance chain set generated by a service instance selection algorithm, includes the number of optional service instance chains, and the total number of optional service instances. Two factors are adjusted and the effects of these factors on the instance chain delay are compared.

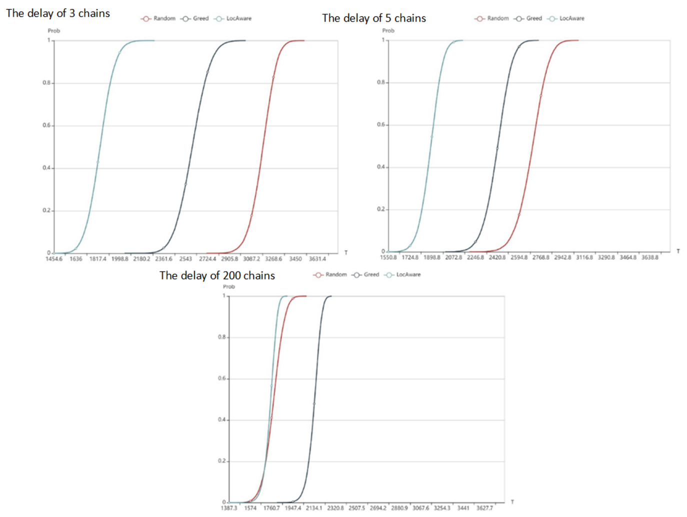

Fig. 5. The first set of experiments.

1# **Experiments.** For the first set of experiments, according to Fig. 5, intuitively, when the number of instance chains is 3 and 5, the performance of the data-aware service composition algorithm is obviously better than the stochastic algorithm and the greedy algorithm. The service delay of the proposed algorithm is shorter, thus reducing the energy consumption of mobile devices. As the number of instance chains increases, the differences between algorithms become more stable, and the performance of each algorithm become better and closer. However, in a real cloud-edge system, it is impossible to allow a service composition algorithm to select a large number of instance chains because the resources at the edge are limited. Therefore, the proposed algorithm still has a good performance improvement on service delay and energy consumption reduction.

2# **Experiments.** As the total number of instances changes, according to Fig. 6, the proposed algorithm is always better than the traditional algorithms in reducing service delay, which means that the proposed algorithm reduced the energy

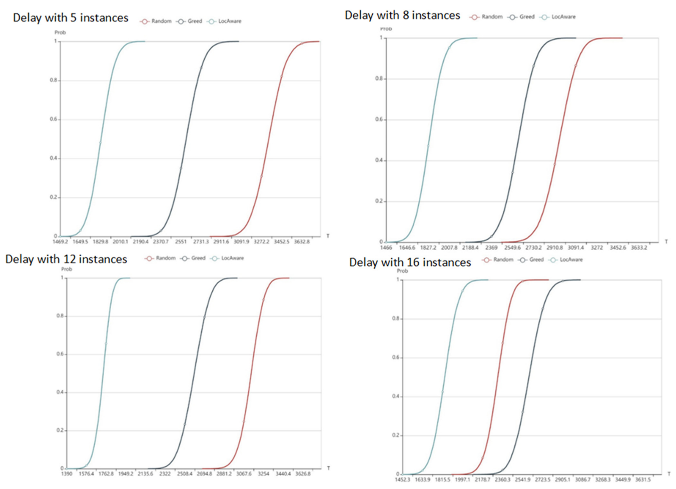

Fig. 6. The second set of experiments.

consumption of mobile devices. At the same time, the performance difference between the three selection algorithms does not change significantly as the number of optional instances increases, because in most cases, the three selection algorithms do not require many chains to select the specified number of instances. Therefore, the proposed algorithm reduces service delay, and it also reduces the waiting time of applications on mobile devices and the energy consumption of mobile devices.

5 Conclusion

In this paper, we investigate service composition problem and propose an energy efficient data-aware service composition algorithm, which is a new extension of service composition in a cloud-edge system. To handle the uncertainty of a single service, we propose an effective service instances selection mechanism to generate multiple service paths to reduce service delay and energy consumption of mobile devices. Extensive simulations show the effectiveness and stability of this method in reducing service delay and energy consumption.

Acknowledgments. This work was supported by science and technology project of State Grid Corporation of China in 2020, Research and Application of Key Technologies of Multiple Data Centers Cooperative Operation and Intelligent Operation and Maintenance for Multi-stite integration, project No. 5210ED200027.

References

1. Hu, Y.C., Patel, M., Sabella, D., Sprecher, N., Young, V.: Mobile edge computing: a key technology towards 5G. ETSI White Paper **11**(11), 1–16 (2015)

2. Paya, A., Marinescu, D.C.: Energy-aware load balancing and application scaling for the cloud ecosystem. IEEE Trans. Cloud Comput. **5**(1), 15–27 (2017)
3. Li, K.: Improving multicore server performance and reducing energy consumption by workload dependent dynamic power management. IEEE Trans. Cloud Comput. **4**(2), 122–137 (2016)
4. Deng, S., Wu, H., Tan, W., Xiang, Z., Wu, Z.: Mobile service selection for composition: an energy consumption perspective. IEEE Trans. Autom. Sci. Eng. **14**(3), 1478–1490 (2017)
5. Gabry, F., Bioglio, V., Land, I.: On energy-efficient edge caching in heterogeneous networks. IEEE J. Sel. Areas Commun. **34**(12), 3288–3298 (2016)
6. Wu, H., Deng, S., Li, W., Fu, M., Yin, J., Zomaya, A.Y.: Service selection for composition in mobile edge computing systems. In: 2018 IEEE International Conference on Web Services (ICWS), San Francisco, CA, pp. 355–358 (2018)
7. Sun, H., Zhou, F., Hu, R.Q.: Joint offloading and computation energy efficiency maximization in a mobile edge computing system. IEEE Trans. Veh. Technol. **68**(3), 3052–3056 (2019)
8. Li, X., Wu, J., Lu, S.: QoS-aware service selection in geographically distributed clouds. In: 2013 22nd International Conference on Computer Communication and Networks (ICCCN), Nassau, pp. 1–5 (2013)
9. Wang, S., Zhou, A., Yang, F., Chang, R.N.: Towards network-aware service composition in the cloud. In: IEEE Transactions on Cloud Computing
10. Wang, S., Zhao, Y., Huang, L., Jinliang, X., Hsu, C.-H.: QoS prediction for service recommendations in mobile edge computing. J. Parallel Distrib. Comput. **127**, 134–144 (2019)
11. Li, X., Lian, Z., Qin, X., Abawajyz, J.: Delay-aware resource allocation for data analysis in cloud-edge system. In: IEEE International Conference on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). Melbourne, Australia, vol. 2018, pp. 816–823 (2018)
12. Ren, J., Guo, H., Xu, C., Zhang, Y.: Serving at the edge: a scalable IoT architecture based on transparent computing. IEEE Netw. **31**(5), 96–105 (2017)
13. Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S., Sabella, D.: On multi-access edge computing: a survey of the emerging 5G network edge cloud architecture and orchestration. In: IEEE Communications Surveys & Tutorials, vol. 19, no. 3, pp. 1657–1681 (2017)
14. Lopez, P.G., et al.: Edge-centric computing: vision and challenges. SIGCOMM Comput. Commun. Rev. **45**(5), 37–42 (2015)