Optimal Application Deployment in Resource Constrained Distributed Edges

Shuiguang Deng, *Senior Member, IEEE*, Zhengzhe Xiang, *Student Member, IEEE*, Javid Taheri, *Member, IEEE*, Mohammad Ali Khoshkholghi, Jianwei Yin, *Member, IEEE*, Albert Y. Zomaya, *Fellow, IEEE*, and Schahram Dustdar, *Fellow, IEEE*

Abstract—The dramatically increasing of mobile applications make it convenient for users to complete complex tasks on their mobile devices. However, the latency brought by unstable wireless networks and the computation failures caused by constrained resources limit the development of mobile computing. A popular approach to solve this problem is to establish a mobile service provisioning system based on a mobile edge computing (MEC) paradigm. In the MEC paradigm, plenty of machines are placed at the edge of the network so that the performance of applications can be optimized by using the involved microservice instances deployed on them. In this paper, we explore the deployment problem of microserivce-based applications in the MEC environment and propose an approach to help to optimize the cost of application deployment with the constraints of resources and the requirement of performance. We conduct a series of experiments to evaluate the performance of our approach. The result shows that our approach can improve the average response time of mobile services.

Index Terms—Mobile service, distributed system, mobile edge computing, service deployment

1 Introduction

We are now embracing an era of mobile computing where about 5.18 million mobile services are serving Chinese mobile users alone.[1] As a result, mobile devices and mobile services play more and more important roles and remolded the communication between people and machines. However, the instability of channels and limited resources of mobile devices prevent users from experiencing high efficiency and seamless interactions with applications. For example, the low computational capability and energy storage [1], [2], [3] of mobile devices restrict the popularization of novel services such as Augmented Reality (AR)/Virtual Reality (VR)/Artificial Intelligence (AI), and the packet losses cause external waiting time for urgent messages. Mobile Edge Computing (MEC) technology is proposed to solve some relevant problems for the aforementioned services [4]. MEC is a novel paradigm that emerges recently as a reinforcement of mobile cloud computing, to optimize the mobile resource usage and wireless network to provide context-aware services [5].

In the MEC paradigm, users can easily connect to the nearby edge servers via wireless network [6] and offload their computation tasks to them. The short-distance connection between users and edge servers can dramatically reduce the latency, and the computation capability of the edge servers are quite qualified to finish those conventional tasks. Additionally, the edge servers do not act alone in many cases–with the help of cluster management techniques, edge servers may coordinate with each other. For example, one edge server can dispatch users' requests to other servers that can handle them. Besides this, the services will also not work alone to fulfill simple tasks–with the help of microservice architecture [7], [8], [9], more complex applications will be easily developed using services in some specific orders. It's a trend that more and more influential IT companies or application vendors start to develop complex applications with microservice techniques (e.g., Kubernetes, Apache Mesos, etc.) nowadays. With this technique, though the developers should be more cautious about the external complexities in application development, the communication controlling and failure recovering, the advantage of decomposing applications into several logically related but functionally individual microservices will bring a high degree of flexibility and reuse and makes it much easier for updating. What's more, it will be much easier to scale out for better performance. And with the help of Container-based techniques, these microservice-based applications can be easily deployed on edge servers.

However, the deployment scheme must be carefully considered, because these servers may have various computation or data storage capacities [10], [11], while the mobile

1. https://aso114.com

- *S. Deng, Z. Xiang, and J. Yin are with the College of Computer Science, Zhejiang University, Hangzhou 310027, China. E-mail: {dengsg, xiangzhengzhe, zjuyjw}@zju.edu.cn.*
- *J. Taheri and M. Khoshkholghi are with the Department of Computer Science, Karlstad University, 651 88 Karlstad, Sweden. E-mail: {javid.taheri, ali.khosh-kholghi}@kau.se.*
- *A.Y. Zomaya is with the School of Computer Science, The University of Sydney, Sydney 2006, Australia. E-mail: albert.zomaya@sydney.edu.au.*
- *S. Dustdar is with Distributed Systems Group, TU Wien, 1040 Vienna, Austria. E-mail: dustdar@infosys.tuwien.ac.at.*

Manuscript received 16 Jan. 2019; revised 1 Nov. 2019; accepted 23 Jan. 2020. Date of publication 30 Jan. 2020; date of current version 2 Apr. 2021. (Corresponding author: Shuiguang Deng.) Digital Object Identifier no. 10.1109/TMC.2020.2970698

users may have different application preferences—if microservices are deployed on servers with low-level hardware or deployed on the edge servers whose connected users rarely use them, the performance of the system will not satisfy both users and vendors. More critically, there would be no doubt that the application vendors can rent lots of edge servers and deploy many instances of microservices to provide better user experience, but the cost of renting edge servers then becomes a major challenge. According to the report of RightScale,[2] a company that specializes in cloud delivery, 26 percent of enterprises with more than 1,000 employees are spending more than 6 million dollars a year on public cloud, but 35 percent of their cloud spendings is wasted—the users may always overrate the resource consumption. On the contrary, the vendors always have clear demand about their applications: they want the applications to keep some key performance indicators (KPIs) [12], e.g., average response time. In this way, we need to work out how to find an appropriate deployment scheme of a microservice-based application with low cost while its actual KPIs are ensured in the service provisioning system based on the MEC paradigm.

In this paper, we mainly focus on the deployment problem of the microservice-based applications with the constraints of application average response time and server resources. The main contributions of this paper are:

1. We highlight the advantages of using MEC architecture to improve the performance of applications.
2. We consider the cooperation of edge servers and the core server in MEC service provisioning systems based on the MEC paradigm, and model the application with a composition of microservices. Based on this, we translate the application deployment problem to the problem of deploying heterogeneous microservice instances.
3. We consider the constraints of resource limitations of edge servers, the business logic of applications as well as the average response time of applications, and then propose an approach to generate appropriate deployment schemes with minimum cost under the on-demand billing model.
4. We conduct a series of experiments to evaluate the performance of the generated deployment schemes and show the improvement compared with other existing baselines.

The rest of this paper is organized as follows. In Section 2 we describe the motivation and scenario of the application deployment problem with an example of a virtual application called $Clairvoyance$. In Section 3 we present how the entities of an MEC service provisioning system work when the microservice-based application is deployed on the system, and give the definition of our problem. In Section 4 we introduce how we formulate this deployment problem to an optimization problem. In Section 5 we describe the details of the approach we proposed to solve this problem. In Section 6 we show the experimental results and analysis about the factors that may affect the results. In Section 7 we highlight related work of edge computing and the corresponding approaches. In Section 8, we conclude our contribution and outline future work.

2. https://www.rightscale.com/lp/state-of-the-cloud

2 Motivation and Scenario

In this section, we will outline the scenario and motivation of our problem with an example. The concept of "Smart City" integrates information and communication technology (ICT), and Internet of things (IoT) to optimize the efficiency of city operations [13], [14], [15], [16]. It allows city officials to interact directly with both community and city infrastructure and to monitor what is happening in the city and how the city is evolving. In smart city projects, one of the most popular topics is smart policing. By deploying webcams, velometers, decibelmeters in the city, illegal behaviors like speeding and unpermitted road work can be easily detected. By equipping the policemen with portable alcometers and ID card readers, lawbreakers will get punished in time. Under this background, assume that an IT company *SoftPoliz*, which devotes itself to help to simplify policing affairs with information technology, has developed an application called $Clairvoyance$. This application aims at providing fast authentication service for policemen so that they can verify criminal suspects effectively. $Clairvoyance$ is made up of 3 related microservices $SC = \{FaceRecognizer, IllegalQuery, AutoAlarm\}$. Besides packing and unpacking the data according to the communication protocol, these 3 microservices have their own function. $FaceRecognizer$ is an image processing service that receives a face image and recognizes the owner, $IllegalQuery$ is data access service which receives ID card number and queries the criminal database with it, $AutoAlarm$ is an alarm service which receives the illegal or criminal records of someone, evaluates the danger level of him (a drug abuser may be not as dangerous as a murder with weapons), and give recommendations about what to do (e.g., wait for reinforcement or arrest on the spot) to policemen according to some laws and cases. By invoking the service in the service chain SC in order, the task will be easily finished. Therefore, the policemen can patrol the city with portable ID card readers and check whether a man is a criminal suspect by taking his photos using the application $Clairvoyance$.

It will be convenient for developers of *SoftPoliz* to deploy the related microservices on a cloud and to invoke them with RESTful APIs [17]. However, better performance is required in this situation, because there are too many people in overcrowded places like railway stations or airports. It is not acceptable to wait minutes for results. A good way to improve the performance is to turn to the MEC architecture. In MEC architectures, the servers in proximity work cooperatively as a platform that integrates the computation and storage capacities of them. With plenty of microservice instances deployed on the distributed nearby edge servers, the latency will be dramatically reduced. According to the experiment in [18], it shows that as much as 72 percent of the communication cost will be saved by taking advantage of MEC architecture in some cases. Fig. 1 shows how it works. In this scenario, every edge server has its own serving area and resource limitation, users in different serving areas will connect to the nearest edge server to invoke the application. The users are not evenly distributed: from

Fig. 1. Smart policing using wearable equipments.

Fig. 1 we can find that the Railway Station and other crowded places included in the serving area of edge server s_1 while s_3 will only serve residential areas. Therefore, more policemen will be assigned to the serving area of s_1, and *Clairvoyance* will be invoked more frequently in the serving area of s_1 than s_3. Intuitively, it will be better to deploy more microservice instances on s_1. However, it is not acceptable for *SoftPoliz* to rent all the resources of edge servers for microservice instances—the cost will be too high to afford. There must be a trade-off between performance and expense: Fig. 2 gives an example of the deployment scheme of application *Clairvoyance*. In this case, there are 3 edge servers and a cloud server (core server). Users invoke the application from different areas, and the related microservices on different servers will be invoked in order to generate the final results. Table 1 shows the configurations of servers in this situation, in which s_0 means the cloud server. The bandwidths (mbps) between the cloud server and edge servers are smaller because of the long-distance communication. And because the core server is a cluster of machines that can easily scale-out, it can have a very large resource capacity. Table 2 shows the parameters of related microservices. When a microservice is deployed on a machine, it will consume some computational and storage resources. There are many types of computational resources like memory and CPU, and we take memory as an example in this case. These microservices may not be homogeneous on different servers, because the operating system and hardware can be different and some special optimization technologies can be adopted on these machines. Therefore, we can find from

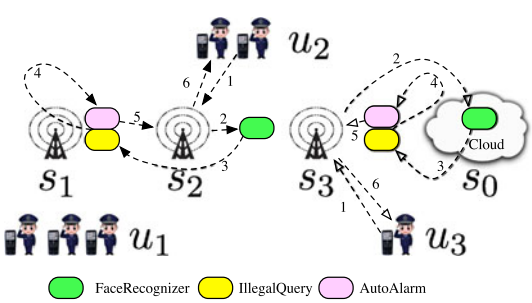

Fig. 2. Using microservice-based applications.

TABLE 1
Configurations of Servers

Server	Bandwidth				Computation capacity	Storage capacity	Wireless rate
	s_0	s_1	s_2	s_3			
s_0	$+\infty$	5	2	4	$+\infty$	$+\infty$	0.6
s_1	4	$+\infty$	20	25	200	400	4
s_2	6	20	$+\infty$	40	200	200	2
s_3	2	10	20	$+\infty$	100	600	4

Table 2 that microservice *FaceRecognizer* will use 10 MB memory and 50 MB disk space, and can process 20 requests per second when it is deployed on edge server s_1. Because the microservices are invoked in order, the output of the previous microservice will be the input of the next microservice. In many cases, resources are charged by the amount of consumption. In this situation, we assume that the price of memory is \$10/MB and the price of the disk is \$25/GB. In addition, we assume that in this time period, the request rates from devices of these three areas can be modeled with Poisson flows [19] whose parameters (request arrival rate) are 20 requests per second, 30 requests per second and 50 requests per second. Then there will be many feasible deployment schemes for the application *Clairvoyance*. For example, the deployment scheme Ω_1 = [[2,0,2,6];[4,0,1,1];[3,3,0,1]] which means deploying 2 instances of *FaceRecognizer* on s_0, 0 instance of *FaceRecognizer* on s_1, 2 instances of *FaceRecognizer* on s_2 and 6 instances of *FaceRecognizer* on s_3 etc.. When we use the scheme Ω_1, the expectation of response time for the application will be 9.56s, and the cost will be \$3872.5. However, if some investigations tell that it is acceptable to wait for less than 12s, a better deployment scheme Ω_2 = [[2,0,2,3];[4,0,1,1];[3,2,0,1]] would be worth considering because the expectation of application response time is 11.15s and the cost can be \$3310.0. Thus, we can find out that it is important to select the deployment scheme carefully.

3 System Model

3.1 Servers and Network

In a typical MEC service provisioning system based on the MEC paradigm, there will be a core server s_0 which acts like the typical cloud platform and n edge servers $s_1, s_2, \ldots, s_n$ distributed in different areas. These servers are available for application developers. Considered as a major form of MEC, mobile base stations (BSs) endowed with cloud-like computing and storage capability are the most common devices that play the role of edge servers [20]. Every edge server s_j has its own serving area and U_j is the set of mobile users in this serving area. The average transmission rate between s_j and users in U_j is v_u^j. These edge servers can cooperate with each other to form a local mobile edge computing platform (sometimes it is named with "Fog Platform") to make full use of their resources. The average bandwidth between the jth edge server and the kth server is $\mathcal{B}_{j,k}$. Especially, because the edge servers can communicate with s_0, the average bandwidth between the core server and the jth edge server is denoted with $\mathcal{B}_{0,j}$ (In general, as the edge servers in a local MEC platform may communicate with each other in a single-hop, $\mathcal{B}_{0,j}$ will always be smaller

TABLE 2
Resource Consumption and Running Parameters of Microservices

Microservice	Computation resource				Storage resource				Processing capacity				Input size	Output size
	s_0	s_1	s_2	s_3	s_0	s_1	s_2	s_3	s_0	s_1	s_2	s_3		
FaceRecognizer	10	10	20	10	50	40	50	50	40	20	30	20	10	1
IllegalQuery	30	30	40	30	30	30	10	30	50	30	40	30	1	5
AutoAlarm	10	10	20	10	60	50	60	60	60	40	50	40	5	2

than $\mathcal{B}_{k,j}$, $k > 0$). The edge server s_j can provide at most L_c^j computation resource and L_d^j storage resource for deploying microservice instances.

3.2 Microservice-Based Application

A microservice ms_i is an abstract concept that describes what task it can complete with specific parameters, it has its own responsibility and scope and can be launched as instances based on container techniques. Receiving a request for ms_i whose average data size is D_i^{in} as input, an instance of ms_i on s_j can process the request with processing capacity $\mu_{i,j}$ and output the result whose average data size is D_i^{out} while it consumes $c_{i,j}$ computation resource and $d_{i,j}$ storage resource on s_j. Here we use the *M/M/c* queue model to describe and evaluate the running of those microservice instances [21], [22], [23], it means there are $\Omega_{i,j}$ workers serving the requests as a queue node $Q_{i,j}$ if there are $\Omega_{i,j}$ instances of microservice ms_i deployed on server s_j. We use this model because that the sojourn time of M/M/c system is less than that of c parallel M/M/1 system. This is easy to prove – if there are k jobs in system, then the M/M/c queue will process with rate = min{k, c}μ while the c paralleling M/M/1 system will process with rate = $j\mu$ where $j \leq$ min{k, c} is the number of active queues. By fulfilling the tasks described by microservices in a service chain $SC = (ms_1, ms_2, \ldots, ms_m)$, the function declared by a microservice-based application is implemented. Though sometimes there will be data access operations in the application which may break the chain structure as shown in Fig. 3, we can also use the following transformation to create a equivalent service chain: In Fig. 4, we can find that microservice ms_2 will access its data ($data_2$). ms_2 first sends query q_2 to tell which part of data it wants, and receives the querying results d_2. The structure (a) can be transformed to (b) by adding two virtual microservices ms_{2s} and ms_{2e}—the input of ms_{2s} is D_1^{out} and the output of ms_{2s} is q_2, the input of ms_{2e} is d_1 and the output of ms_{2e} is D_2^{out}. The data microservice then becomes the successor of ms_{2s} and the predecessor of ms_{2e} in the service chain. Here the resource consumption of ms_{2s} is the same as ms_2, but ms_{2e} will not consume any resources. What's more, ms_{2s} and ms_{2e} will share the same deployment scheme as ms_2. In this way, we only focus on the chain structure, which means that the input of the application is the input of ms_1 and the output of ms_m is the required output for the application. Besides these, ms_{i+1} will use the output of ms_i as input.

3.3 Request Life Cycle

Denote the probability that server s_j dispatch requests about ms_i to s_k with $Pr_{j,k}^i$, which describes the routing policy, we can overview the life cycle of a request in Fig. 5: For u in U_j, when he/she tries to use application described by SC, his/her device will first produce a request with input in_1 about it and send the request to s_j. According to the probabilities $Pr_{j,*}^1$, this request is sent to server s_{k_1} to fulfill the task declared in ms_1. The instances of ms_1 on s_{k_1} finish this task and get the output out_1, then produce a new request whose input $in_2 = out_1$ and send it to server s_{k_2} according to the probabilities $Pr_{k_1,*}^2$. Step by step, the instances of ms_m on s_{k_m} finally get the output out_m, which is the result of the application. The final result will be sent back to s_j and then to u via s_j.

As requests produced by U_j will be dynamic in one day, here we follow the works in [20], [24] etc. to divide time into discrete time periods (or time slots) in which the requests of U_j in time period t_p can be modeled with a Poisson flow whose average request arrival rate is $\lambda_j^{t_p}$ (some techniques like [25] may be used to help to predict). In every time period, the deployment scheme can be updated. The length of a time period is not fixed, but it won't be long so that the system won't update frequently. Therefore, the deployment problem over time is divided into a series of service deployment subproblems over time periods. In the rest of this paper, we will omit the superscript t_p of $\lambda_j^{t_p}$ (namely, we will use λ_j) and focus on the service deployment scheme in one time period.

3.4 Billing Model

Different companies have their own billing models. For example, there are two types of billing models for Amazon

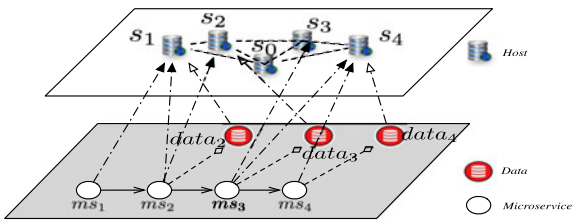

Fig. 3. An example of application deployment.

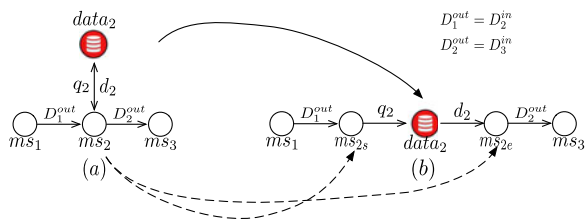

Fig. 4. An example of equivalent structure transformation.

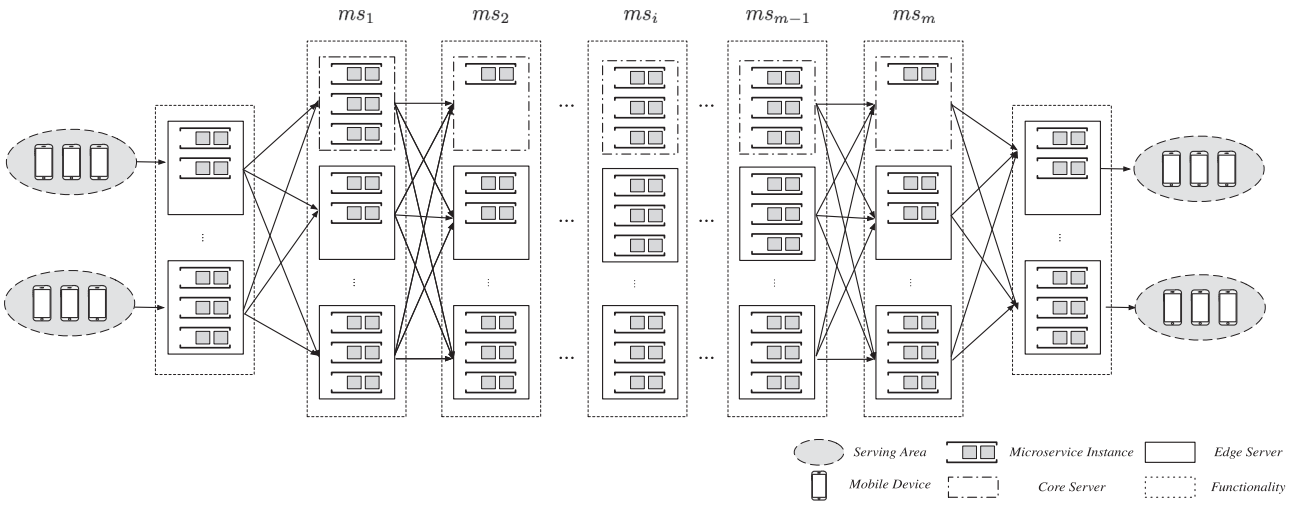

Fig. 5. An overview of the MEC service provisioning system.

Elastic Container Service:[3] the Fargate launch type model and the EC2 launch type model. With the Fargate model, you pay for the amount of vCPU and memory resources that your containerized application requests and you pay for AWS resources you create to store and run your application in the EC2 model. In this work, we mainly consider the on-demand billing in evaluating the cost of the deployment scheme—it means that the more resource is used, the more you have to pay. Without loss of generation, here we assume that the cost is proportional to the used resource, and the unit cost of computation resource and storage resource are represented by α and β respectively.

3.5 Problem Definition

With the introduction of related concepts, now we can give the definition of the *Optimal Instance Deployment Problem* (OIDP) clearly: Given the core server and edge servers $\mathcal{S} = \{s_0, s_1, \ldots, s_n\}$, an application $\mathcal{A}$ whose service chain is $\mathcal{SC} = \{ms_1, ms_2, \ldots, ms_m\}$, and users' average request rate for $\mathcal{A}$ on different edge servers represented with $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_n)^{\mathrm{T}}$ in a time period, find the deployment scheme $\boldsymbol{\Omega} = \{\Omega_{i,j}\}_{i=1,j=0}^{m,n}$ with minimum cost so that the application can serve the users with an average response time no more than T^*.

4 Problem Formulation

In this section, we will clarify the objective and constraints of the problem and formulate them in a brief way.

4.1 Objective of Deployment Problem

In this work, we mainly consider the computation cost and storage cost of microservices. According to the explanation of billing model in Section 3, the cost of resource consumption can be represented as

$$C(\Omega) = \alpha \sum_{i=0}^{m} \sum_{j=0}^{n} c_{i,j}\Omega_{i,j} + \beta \sum_{i=0}^{m} \sum_{j=0}^{n} d_{i,j}\Omega_{i,j}. \tag{1}$$

If we denote $\gamma_{i,j} \triangleq \alpha c_{i,j} + \beta d_{i,j}$ as the cost of instances of different microservices. By vectorize $\gamma_{i,j}$ and $\Omega_{i,j}$ with the order of service chain, we can get two column vectors $\boldsymbol{\gamma} = (\gamma_{1,0}, \ldots, \gamma_{1,n}, \ldots, \gamma_{m,0}, \ldots, \gamma_{m,n})^{\mathsf{T}}$ and $\boldsymbol{\Omega} = (\Omega_{1,0}, \ldots, \Omega_{1,n}, \ldots, \Omega_{m,0}, \ldots, \Omega_{m,n})^{\mathsf{T}}$ whose dimension θ is $m * (n+1)$. Then the cost $C(\boldsymbol{\Omega})$ can be represented as

$$C(\boldsymbol{\Omega}) = \boldsymbol{\gamma}^{\mathsf{T}}\boldsymbol{\Omega}. \tag{2}$$

4.2 Constraint of Application Response Time

Here we denote $\phi = (\phi_s, \phi_1, \phi_2, \ldots, \phi_m, \phi_e)$ as the request path to describe a request's life cycle, it shows the order of hosts to handle this request. Denote P_ϕ the probability of request path ϕ, and T_ϕ is the total time for requests that go through path ϕ. The average application response time can be represented as

$$\mathbb{E}[T] = \sum_{\phi_s=1}^{n} \underbrace{\sum_{\phi_1=0}^{n} \cdots \sum_{\phi_m=0}^{n}}_{\mathrm{m}} \sum_{\phi_e=1}^{n} P_{\boldsymbol{\phi}} T_{\boldsymbol{\phi}}. \tag{3}$$

In this way, we will investigate P_ϕ and T_ϕ respectively to calculate $\mathbb{E}[T]$.

4.2.1 Probability of Request Path

With the definition of $Pr_{i,j}^t$, we can represent P_ϕ as

$$P_\phi = Pr_{\phi_s} Pr_{\phi_s,\phi_1}^s \left(\prod_{t=1}^{m-1} Pr_{\phi_t,\phi_{t+1}}^t \right) Pr_{\phi_m,\phi_e}^e. \tag{4}$$

Here Pr_{ϕ_s} means the probability that the nearby edge server is s_{ϕ_s}. Because the requests will always go back to the caller and his nearby edge server, $Pr_{\phi_m\phi_e}^e$ will not effect the value of P_ϕ. Thus, we have

$$P_\phi = Pr_{\phi_s} Pr_{\phi_s,\phi_1}^s \left(\prod_{t=1}^{m-1} Pr_{\phi_t,\phi_{t+1}}^t \right). \tag{5}$$

P_ϕ will be different under different service routing polices. There are many reasonable routing polices because different developers may consider different factors. For example:

3. https://aws.amazon.com/ecs/pricing/?nc1=h_ls

Round-Robin. Under this policy, the instances of a microservice on different servers will have the same probability to receive requests—if ms_1 has 1 instance on s_1 and has 2 instances on s_2, the probability that request of ms_1 goes to s_2 is twice as much as that to s_1.

Weighted Routing. Under this policy, the processing capability is considered as another factors that can help scheduling requests, the probability a instance of microservices receives is proportional to processing capability—if ms_1 has 1 instance on s_1 whose processing capability is 200 request/sec and has 2 instances on s_2 whose processing capability is 100 request/sec, the probability that request of ms_1 goes to s_2 is the same as that to s_1, because $1 \times 200 = 2 \times 100$.

In this work, we will take the round-robin policy as example to explain how we will formulate the deployment problem, so that developers can easily follow the process with their own routing policies.

It is obvious that Pr_{ϕ_s} is dependent on the distribution of application requests, therefore we have

$$Pr_{\phi_s} = \frac{\lambda_{\phi_s}}{\sum_{i=1}^{n} \lambda_i}, \tag{6}$$

because the requests will be dispatched to instances according to the amount in round-robin policy, we have

$$Pr^s_{\phi_s,\phi_1} = \frac{\Omega_{1,\phi_1}}{\sum_{k=0}^{n} \Omega_{1,k}}, Pr^t_{\phi_t,\phi_{t+1}} = \frac{\Omega_{t+1,\phi_{t+1}}}{\sum_{k=0}^{n} \Omega_{t+1,k}}. \tag{7}$$

4.2.2 Response Time of Request Path

For each $T_{\boldsymbol{\phi}}$, it includes the access time, routing time, queue time and backhaul time

$$T_{\boldsymbol{\phi}} = T_{access} + T_{routing} + T_{queue} + T_{backhaul}. \tag{8}$$

Where the four parts can be computed as follows:

a) Access Time. The access time has two parts, the transmission time between mobile devices to their nearby edge server s_{ϕ_s} and the transmission time from s_{ϕ_s} to server s_{ϕ_1} which caches the instances of ms_1. Therefore, the access time is

$$T_{access} = \frac{D^{in}_1}{v^{\phi_s}_u} + \frac{D^{in}_1}{\mathcal{B}_{\phi_s\phi_1}}. \tag{9}$$

b) Routing Time. When any instance has finished its work, the result will be routed to the next microservice instance. Therefore, the routing time can be represented as

$$T_{routing} = \sum_{i=1}^{m-1} \frac{D^{out}_i}{\mathcal{B}_{\phi_i\phi_{i+1}}}. \tag{10}$$

c) Queue Time. The queue time includes the execution time and waiting time. Given the processing capacity μ_{i,ϕ_i}, the execution time T^e_{i,ϕ_i} can be represented as

$$T^e_{i,\phi_i} = \frac{1}{\mu_{i,\phi_i}}. \tag{11}$$

At the same time, we use T^w_{i,ϕ_i} to denote the expectation of waiting time in the queue of ms_i's instance on server s_{ϕ_i}. According to the queuing theory, T^w_{i,ϕ_i} can be represented as

$$T^w_{i,\phi_i} = \frac{1/\mu_{i,\phi_i}}{\Omega_{i,\phi_i}(1-\rho_{i,\phi_i})[1+(1-\rho_{i,\phi_i})\Upsilon_{i,\phi_i}]}, \tag{12}$$

where $\Upsilon_{i,\phi_i} \triangleq \frac{\Omega_{i,\phi_i}!}{(\Omega_{i,\phi_i}\rho_{i,\phi_i})^{\Omega_{i,\phi_i}}} \sum_{k=0}^{\Omega_{i,\phi_i}-1} \frac{(\Omega_{i,\phi_i}\rho_{i,\phi_i})^k}{k!}$ is used here to simplify the expression.

Note that there is a parameter $\rho_{i,j}$ involved in the expression of T^w_{i,ϕ_i}. It means the serving utilization of queuing node $Q_{i,j}$. And according to the queuing theory, $\rho_{i,j}$ can be represented as

$$\rho_{i,j} = \frac{\lambda'_{i,j}}{\mu'_{i,j}}, \tag{13}$$

here $\lambda'_{i,j}$ is the average request arrival rate of for microservice instances ms_i at node $Q_{i,j}$, and $\mu'_{i,j} = \Omega_{i,j}\mu_{i,j}$ is the processing rate. $\rho_{i,j}$ is always less than 1 so that requests will not be blocked in the queuing node.

Suppose $\boldsymbol{\lambda}'_{i+1} = (\lambda'_{i+1,0}, \lambda'_{i+1,1}, \ldots, \lambda'_{i+1,n})^{\intercal}$ is the request arrival rates for instances on different severs of ms_{i+1}. According to Burke's theorem [19], the request leaving rates for instances on different severs of ms_i will be equal to $\boldsymbol{\lambda}'_{i+1}$. Denote $\mathbf{Pr}^i$ as the routing matrix for requests generated from ms_i

$$\mathbf{Pr}^i \triangleq \begin{bmatrix} Pr^i_{0,0} & Pr^i_{1,0} & \cdots & Pr^i_{n,0} \\ Pr^i_{0,1} & Pr^i_{1,1} & \cdots & Pr^i_{n,1} \\ \vdots & \vdots & \ddots & \vdots \\ Pr^i_{0,n} & Pr^i_{1,n} & \cdots & Pr^i_{n,n} \end{bmatrix}. \tag{14}$$

As the microservices are invoked one by one, the requests will go to the next microservice instances when previous tasks are fulfilled. Therefore, we can use the following equation to describe the relation between $\boldsymbol{\lambda}'_{i+1}$ and $\boldsymbol{\lambda}'_{i}$:

$$\boldsymbol{\lambda}'_{i+1} = \mathbf{Pr}^i \boldsymbol{\lambda}'_i, \tag{15}$$

while the elements of $\boldsymbol{\lambda}'_1$ are initialized with

$$\lambda'_{1,j} = \sum_{k=1}^{n} Pr^s_{k,j}\lambda_k. \tag{16}$$

By solving the Equation (15), we can get

$$\lambda'_{i,j} = \frac{\Omega_{i,j}}{\sum_{k=0}^{n} \Omega_{i,k}} \sum_{k=1}^{n} \lambda_k. \tag{17}$$

Therefore, the sojourn time in queue is

$$T_{queue} = \sum_{i=1}^{m} (T^e_{i,\phi_i} + T^w_{i,\phi_i}). \tag{18}$$

d) Backhaul Time. The backhaul time has two parts as well—the transmission time between the edge server s_{ϕ_e} to connected mobile devices and the transmission time from s_{ϕ_m} to server s_{ϕ_e} ($\phi_e = \phi_s$). Therefore, the backhaul time is

$$T_{backhaul} = \frac{D^{out}_m}{v^{\phi_e}_u} + \frac{D^{out}_m}{\mathcal{B}_{\phi_m\phi_e}}. \tag{19}$$

4.2.3 Application Response Time Estimation

As the response time of a request path is divided into four parts, and because $\mathbb{E}[X+Y] = \mathbb{E}[X] + \mathbb{E}[Y]$, the expectation

of application response time $\mathbb{E}[T]$ can be represented by the sum of $\mathbb{E}[T_{access}]$, $\mathbb{E}[T_{backhaul}]$, $\mathbb{E}[T_{routing}]$ and $\mathbb{E}[T_{queue}]$. With the former equations, we can get these time costs separately, and substitute Eqs. ((5), (6), (7), (9), (10), (18), (19)) into Eq. (3), the expectation of application response time can be represented as follows (with the help of auxiliary variables shown in Appendix B, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TMC.2020.2970698)

$$\mathbb{E}[T] = \kappa\left(\boldsymbol{\lambda}^{\intercal}\mathbf{v}_u^{\oslash} + \frac{\boldsymbol{\lambda}^{\intercal}\mathbf{H}\boldsymbol{\Omega}}{\boldsymbol{e}_1^{\intercal}\boldsymbol{\Omega}}\right) + \sum_{i=1}^{m-1}\frac{\boldsymbol{\Omega}^{\intercal}\mathbf{W}_i\boldsymbol{\Omega}}{\boldsymbol{\Omega}^{\intercal}J_i\boldsymbol{\Omega}} + \sum_{i=1}^{m}\frac{\boldsymbol{\eta}_i^{\intercal}\boldsymbol{\Omega}}{\boldsymbol{e}_i^{\intercal}\boldsymbol{\Omega}}. \tag{20}$$

As a consequence, when rewrite $\mathbb{E}[T]$ with $\mathbb{E}[T(\boldsymbol{\Omega})]$ for the decision variable $\boldsymbol{\Omega}$, the constraint of application response time can be represented as

$$\mathbb{E}[T(\boldsymbol{\Omega})] \leq T^*. \tag{21}$$

4.3 Constraint of Resource Consumption

Though edge servers are powerful machines with larger storage and faster computation units, they have limitations on their resources. Besides this, there are still many other application to be deployed, it is not possible to give all the resource to an specific application. Here we use $\boldsymbol{L_c} = (L_c^0, L_c^1, \ldots, L_c^n)^{\intercal}$ and $\boldsymbol{L_d} = (L_d^0, L_d^1, \ldots, L_d^n)^{\intercal}$ to represent the computation resource quota and storage resource quota for application $\mathcal{A}$. By denoting $\mathbf{C_R}$ the constraint matrix of resources, and $\mathbf{L}$ the concatenation of $\boldsymbol{L_c}$ and $\boldsymbol{L_d}$,

$$\mathbf{C_R} \triangleq \begin{bmatrix} c_{1,0} & \ldots & 0 & \ldots & \ldots & c_{m,0} & \ldots & 0 \\ \vdots & \ddots & \vdots & & & \vdots & \ddots & \vdots \\ 0 & \ldots & c_{1,n} & \ldots & \ldots & 0 & \ldots & c_{m,n} \\ d_{1,0} & \ldots & 0 & \ldots & \ldots & d_{m,0} & \ldots & 0 \\ \vdots & \ddots & \vdots & & & \vdots & \ddots & \vdots \\ 0 & \ldots & d_{1,n} & \ldots & \ldots & 0 & \ldots & d_{m,n} \end{bmatrix},$$

then we can describe the constraint of edge resources as

$$\mathbf{C_R}\boldsymbol{\Omega} \leq \mathbf{L}. \tag{22}$$

4.4 Constraint of Business Logic

As the microservices works in order to fulfill complex tasks, the absence of microservice instance for any microservice in the service chain SC will not be allowed. Therefore, we have

$$-\mathbf{C_B}\boldsymbol{\Omega} \leq -\vec{\mathbf{1}}. \tag{23}$$

Here the business logic constraint matrix $\mathbf{C_B}$ is denoted by

$$\mathbf{C_B} \triangleq \begin{bmatrix} 1 & \ldots & 1 & 0 & \ldots & 0 & \ldots & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 1 & \ldots & 1 & \ldots & 0 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ldots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0 & 0 & \ldots & 0 & \ldots & 1 & \ldots & 1 \end{bmatrix},$$

and $\vec{\mathbf{1}} = (1, 1, \ldots, 1)^{\intercal}$.

4.5 Constraint of Queuing System

In queuing system, the parameter ρ means the serving utilization of the queuing node. As mentioned in Section 4.2.2, ρ should be positive and less than 1.0 for a stable queuing system. Otherwise, the requests will heap up so that the system cannot handle them anymore. With Eq. (12), we can represent the constraint by

$$-\mathbf{C_Q}\boldsymbol{\Omega} \leq -\Lambda \cdot \vec{\mathbf{1}}, \tag{24}$$

where the constraint matrix of queue system $\mathbf{C_Q}$ is denoted with

$$\mathbf{C_Q} \triangleq \begin{bmatrix} \min_j \mu_{1,j} & \ldots & \min_j \mu_{1,j} & \ldots & 0 & \ldots & 0 \\ 0 & \ldots & 0 & \cdots & 0 & \ldots & 0 \\ \vdots & \ddots & \vdots & \ldots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0 & \ldots & \min_j \mu_{m,j} & \ldots & \min_j \mu_{m,j} \end{bmatrix},$$

and $\Lambda = \sum_n^{k=1} \lambda_k$ is the total request arrival rate.

5 Approaches

With the objective and constraints shown in Section.6, we now can easily the formulate the OIDP as the following optimization problem:

$$P_1 : \min\ \boldsymbol{\gamma}^{\intercal}\boldsymbol{\Omega}$$
$$s.t. \begin{cases} \kappa(\boldsymbol{\lambda}^{\intercal}\mathbf{v}_u^{\oslash} + \frac{\boldsymbol{\lambda}^{\intercal}\mathbf{H}\boldsymbol{\Omega}}{\boldsymbol{e}_1^{\intercal}\boldsymbol{\Omega}}) + \sum_{i=1}^{m-1}\frac{\boldsymbol{\Omega}^{\intercal}\mathbf{W}_i\boldsymbol{\Omega}}{\boldsymbol{\Omega}^{\intercal}J_i\boldsymbol{\Omega}} + \sum_{i=1}^{m}\frac{\boldsymbol{\eta}_i^{\intercal}\boldsymbol{\Omega}}{\boldsymbol{e}_i^{\intercal}\boldsymbol{\Omega}} \leq T^* \\ \mathbf{A}\boldsymbol{\Omega} \leq \mathbf{b}, \boldsymbol{\Omega} \in \mathbb{N}^{\theta} \end{cases}, \tag{25}$$

where $\mathbf{A} \triangleq (\mathbf{C_R}, -\mathbf{C_B}, -\mathbf{C_Q})^{\intercal}$ and $\mathbf{b} \triangleq (\mathbf{L}, -\vec{\mathbf{1}}, -\Lambda \cdot \vec{\mathbf{1}})^{\intercal}$. The definitions of the auxiliary variables $\mathbf{v}_u^{\oslash}$, κ, $\boldsymbol{e}_i$, $\boldsymbol{\eta}_i$, $\mathbf{H}$, $\mathbf{W}_i$ and $\mathbf{J}_i$ are shown in Appendix B, available in the online supplemental material.

Therefore, searching the optimal application deployment scheme is to find the scheme $\boldsymbol{\Omega}^*$ from the feasible region which has the minimum cost $\boldsymbol{\gamma}^{\intercal}\boldsymbol{\Omega}^*$. From the form of P_1, we can find that it is a nonlinear integer programming problem, which is NP-Complete. Therefore, we turn to approaches that can help to find some suboptimums. At first, we will relax the constraint of $\mathbb{N}$ to $\mathbb{R}_0$ ($\mathbb{R}_0 = \mathbb{R} - \mathbb{R}^+$) so that we can take advantage of the optimization technology for continuous problems. And then, the branch and bound technique will be adopted to find the integer solutions.

However, as the queue time vector η_i is derived from the queueing theory, in which the waiting time $T_{i,j}^w$ is calculated by summation of sequence (in $\Upsilon_{i,j}$). It will be hard for us to compute the derivatives. Therefore, we need a continuous and easier form of $T_{i,j}^w$, and if it is an approximation form, we should quantify the gap between it and the original one. By setting $\rho = \rho_{i,\phi_i}$ and $n = \Omega_{i,\phi_i}$ for the 3rd Lemma shown in Appendix A, available in the online supplemental material, we will have the approximation form of Υ_{i,ϕ_i}

$$\hat{\Upsilon}_{i,\phi_i} = e^{-(1-\rho_{i,\phi_i})(\Omega_{i,\phi_i}-1)}\rho_{i,\phi_i}{}^{-\Omega_{i,\phi_i}}, \tag{26}$$

and $\Upsilon_{i,\phi_i} \leq \hat{\Upsilon}_{i,\phi_i}$. Then the upper bound $\widehat{T}_{i,\phi_i}^w$ of waiting time in queue is

$$\widehat{T}_{i,\phi_i}^w = \frac{1}{\mu_{i,\phi_i}\Omega_{i,j}}\left(\frac{1}{1-\rho_{i,\phi_i}} - \frac{1}{1-\rho_{i,\phi_i}+\hat{\Upsilon}_{i,\phi_i}^{-1}}\right). \tag{27}$$

Similarly, we can also get that

$$\frac{1}{1-\rho_{i,\phi_i}+\Upsilon_{i,\phi_i}^{-1}}-\frac{1}{1-\rho_{i,\phi_i}+\hat{\Upsilon}_{i,\phi_i}^{-1}}\leq\Omega_{i,\phi_i}. \tag{28}$$

Therefore, the relation between $T_{i,j}^{w}$ and $\widehat{T}_{i,j}^{w}$ will be clear

$$\widehat{T}_{i,j}^{w}-T_{i,j}^{w}\ \leq\frac{1}{\mu_{i,j}\Omega_{i,j}}\cdot\Omega_{i,j}=\frac{1}{\mu_{i,j}}. \tag{29}$$

On the other hand, for all positive vector $\boldsymbol{x}\in\mathbb{R}^{\theta}$ we have

$$\min_{q\in[1,\theta]}\omega_q\leq\frac{\omega^{\intercal}\boldsymbol{x}}{\boldsymbol{1}\cdot\boldsymbol{x}}\leq\max_{q\in[1,\theta]}\omega_q. \tag{30}$$

Therefore, by carefully choosing a parameter $\epsilon\in[0,\frac{1}{\mu_{min}}]$ ($\mu_{min}=\min_{i,j}\mu_{i,j}$), we can transform the constraint of $\mathbb{E}[T(\boldsymbol{\Omega})]$ to the following equivalent one:

$$\mathbb{E}[\hat{T}(\boldsymbol{\Omega})]-\epsilon\leq T^{*}, \tag{31}$$

where the $\mathbb{E}[\hat{T}(\boldsymbol{\Omega})]$ has the same structure as Eq. (20), except that the queue time vector η_i in $\mathbb{E}[\hat{T}(\boldsymbol{\Omega})]$ is replaced with

$$\boldsymbol{\eta_i}\triangleq(\underbrace{0,\ldots,0}_{(i-1)(n+1)},\frac{1}{\mu_{i,0}}+\hat{T}_{i,0}^{w},\ldots,\frac{1}{\mu_{i,n}}+\hat{T}_{i,n}^{w},\underbrace{0,\ldots,0}_{(m-i)(n+1)})^{\intercal}. \tag{32}$$

Suppose b_k is the kth element of vector $\boldsymbol{b}$ and $\boldsymbol{A}_k$ is the kth column vector of $\boldsymbol{A}$, and denote the constraints with

$$\begin{aligned}c_0(\boldsymbol{\Omega})&=\epsilon+T^{*}-\mathbb{E}[\hat{T}(\boldsymbol{\Omega})]\\ c_k(\boldsymbol{\Omega})&=b_k-\boldsymbol{A}_k\boldsymbol{\Omega},\end{aligned} \tag{33}$$

then we can minimize an l_1-penalty function with some sufficiently-large penalty factor ν to solve $\boldsymbol{P}_1$ [26]

$$\min_{\boldsymbol{\Omega}\in\mathbb{R}^{\theta}}\Psi(\boldsymbol{\Omega};\nu)=C(\boldsymbol{\Omega})+\nu\sum_{k}\max(-c_k(\boldsymbol{\Omega}),0). \tag{34}$$

What's more, by smoothing this penalty function with some elastic variables $\boldsymbol{w}$ and regarding the concatenated vector $\boldsymbol{x}_P=(\boldsymbol{\Omega},\boldsymbol{w})$ as points in the expanded space, we can get problem P_1's smooth version (P_2)

$$\begin{aligned}\boldsymbol{P}_2:\ &\min_{\boldsymbol{\Omega}\in\mathbb{R}^{\theta}}\Psi^{S}(\boldsymbol{\Omega},\boldsymbol{w};\nu)=C(\boldsymbol{\Omega})+\nu\sum_{k}w_k\\ &s.t.\qquad c_k(\boldsymbol{\Omega})+w_k\geq0,w_k\geq0.\end{aligned} \tag{35}$$

Thus we can now apply the primal-dual interior point method [27] to find the suboptimal of $\boldsymbol{P}_2$. Namely, we need to solve a sequence of unconstrained problems ($\boldsymbol{Q}_t$)

$$\boldsymbol{Q}_t:\min_{(\boldsymbol{\Omega},\boldsymbol{w})}\Psi^{B}(\boldsymbol{\Omega},\boldsymbol{w};\tau^{t},\nu), \tag{36}$$

where the objective $\Psi^{B}(\boldsymbol{\Omega},\boldsymbol{w};\tau^{t},\nu)$ is represented with the following logarithmic barrier form:

$$\begin{aligned}\Psi^{B}(\boldsymbol{\Omega},\boldsymbol{w};\tau^{t},\nu)=\Psi^{S}(\boldsymbol{\Omega},\boldsymbol{w};\nu)&-\tau^{t}\sum_{k}\log(w_k)\\ &-\tau^{t}\sum_{k}\log(c_k(\boldsymbol{\Omega})+w_k),\end{aligned} \tag{37}$$

ν is the penalty factor that measures the infeasibility of subproblem $\boldsymbol{Q}_t$ and τ^{t} is the barrier factor that manages the constraints shown in $\boldsymbol{P}_2$. According to the requirement of the primal-dual interior point method [27], $\{\tau\}$ should be a decreasing sequence where $\lim_{t\to\infty}=0$, and the minimizer $(\boldsymbol{\Omega}_{t+1}^{*},\boldsymbol{w}_{t+1}^{*})$ will be generated by solving $\boldsymbol{Q}_t$ with $\boldsymbol{Q}_t$'s minimizer $(\boldsymbol{\Omega}_{t}^{*},\boldsymbol{w}_{t}^{*})$ as initial point. And the minimizer will finally converge to the minimizer of $\boldsymbol{P}_2$ (see in Section 5.1). It's worth noting that with the elastic variable $\boldsymbol{w}$, the initial feasible point for the interior point method will be easily got by setting $w_k\geq\max(-c_k(\boldsymbol{\Omega}),0)$ for any $\boldsymbol{\Omega}$.

Before solving the problem $\boldsymbol{Q}_t$, we will first denote the primal-dual function $\Phi(\boldsymbol{x}_P,\boldsymbol{x}_D;\tau^{t},\nu)$ for it. Suppose the primal first-order Lagrange multiplier estimates are denoted with

$$\begin{aligned}\boldsymbol{y}&\triangleq\tau^{t}(\boldsymbol{C}_{diag}(\boldsymbol{\Omega})+\boldsymbol{W}_{diag})^{-1}\vec{\boldsymbol{1}}\\ \boldsymbol{u}&\triangleq\tau^{t}(\boldsymbol{W}_{diag})^{-1}\vec{\boldsymbol{1}},\end{aligned} \tag{38}$$

where we use vectors $\boldsymbol{c}(\boldsymbol{\Omega})$ to represent the above constraints for convenience, $\boldsymbol{C}_{diag}(\boldsymbol{\Omega})$ and $\boldsymbol{W}_{diag}$ are matrices that diagonalized from $\boldsymbol{c}(\boldsymbol{\Omega})$ and $\boldsymbol{w}$. Then the primal-dual function for primal vector $\boldsymbol{x}_P=(\boldsymbol{\Omega},\boldsymbol{w})$ and dual vector $\boldsymbol{x}_D=(\boldsymbol{y},\boldsymbol{u})$ can be represented with

$$\Phi(\boldsymbol{x}_P,\boldsymbol{x}_D;\tau^{t},\nu)\triangleq\begin{bmatrix}\boldsymbol{\gamma}-J^{\intercal}(\boldsymbol{\Omega})\boldsymbol{y}\\ \nu-\boldsymbol{y}-\boldsymbol{u}\\ (\boldsymbol{C}_{diag}(\boldsymbol{\Omega})+\boldsymbol{W}_{diag})\boldsymbol{y}-\tau^{t}\\ \boldsymbol{W}_{diag}\boldsymbol{u}-\tau^{t}\end{bmatrix}, \tag{39}$$

here $J(\boldsymbol{\Omega})$ is the Jacobian matrix of $\boldsymbol{c}(\boldsymbol{\Omega})$, and the sum of vector $\boldsymbol{x}$ and scalar a here means that every elements of $\boldsymbol{x}$ is added with a for convenience. Then the Karush-Kuhn-Tucker (KKT) condition can be represented with

$$\begin{aligned}\Phi(\boldsymbol{x}_P,\boldsymbol{x}_D;\tau^{t},\nu)&=\boldsymbol{0}\\ (\boldsymbol{c}(\boldsymbol{\Omega})+\boldsymbol{w},\boldsymbol{w},\boldsymbol{y},\boldsymbol{u})&\geq\boldsymbol{0}.\end{aligned} \tag{40}$$

As $\lim_{t\to\infty}\tau^{t}=0$, and the KKT condition for $\boldsymbol{P}_2$ can be represented with $\Phi(\boldsymbol{x}_P,\boldsymbol{x}_D;0,\nu)$, then we can find that if there is a limit point $\boldsymbol{x}^{*}=(\boldsymbol{\Omega}^{*},\boldsymbol{w}^{*},\boldsymbol{y}^{*},\boldsymbol{u}^{*})$ generated by solving $\boldsymbol{Q}_t$ when t is sufficiently large, it will be a KKT point for $\boldsymbol{P}_2$. And because $\boldsymbol{P}_2$ is equivalent to $\boldsymbol{P}_1$, it will also be the KKT point for $\boldsymbol{P}_1$.

As with the process introduced in [28], the goal to solve the problem $\boldsymbol{Q}_t$ is to generate appropriate $(\boldsymbol{x}_P^{t+1},\boldsymbol{x}_D^{t+1})$. Suppose $\delta=(\delta_P,\delta_D)$ is the direction vector so that $(\boldsymbol{x}_P^{t+1},\boldsymbol{x}_D^{t+1})=(\boldsymbol{x}_P^{t+1}+\delta_P,\boldsymbol{x}_D^{t+1}+\delta_D)$, then δ can be approximated by

$$\min_{\boldsymbol{\delta}}\boldsymbol{\delta}^{\intercal}\Phi(\boldsymbol{x}_P^{t},\boldsymbol{x}_D^{t};\tau^{t},\nu)+\frac{1}{2}\boldsymbol{\delta}^{\intercal}\nabla\Phi(\boldsymbol{x}_P^{t},\boldsymbol{x}_D^{t};\tau^{t},\nu)\boldsymbol{\delta}. \tag{41}$$

Thus, the solution of the following linear system

$$\nabla_{(\boldsymbol{x}_P^{t},\boldsymbol{x}_D^{t})}\Phi(\boldsymbol{x}_P^{t},\boldsymbol{x}_D^{t};\tau^{t},\nu)\boldsymbol{\delta}=-\Phi(\boldsymbol{x}_P^{t},\boldsymbol{x}_D^{t};\tau^{t},\nu), \tag{42}$$

can be used as the direction vector [29].

With the above statements, the framework of the primal-dual interior point algorithm will be clear: iteratively find the by approximately solving $\boldsymbol{Q}_t$ until τ^{t} is small enough and $||\Phi(\boldsymbol{x}_P,\boldsymbol{x}_D;\tau^{t},\nu)||$ less or equal than some tolerance, the detail of it is shown in Algorithm 1, whose name is

ID4AReE. It is named after **I**nstance **D**eployment **A**pproximation algorithm for **Re**source constrained **E**dges.

Algorithm 1. Instance Deployment Approximation Algorithm for Resource constrained Edges, IDA4ReE

Input:
$\boldsymbol{\gamma}$: the cost vector;
$\boldsymbol{c}(\cdot)$: the constriant functions;
τ: the initial barrier factor, $\tau \in (0,1)$
ν: the penalty factor, $\nu > 0$
Output:
$\boldsymbol{\Omega}^*$: the deployment scheme of instances;
Initialize $\boldsymbol{\Omega}^0 \in \mathbb{R}^\theta$
Initialize $\boldsymbol{w}^0 \in \mathbb{R}_+^K$ so that $\boldsymbol{c}(\boldsymbol{\Omega}^0) + \boldsymbol{w}^0 > \mathbf{0}$
Initialize dual estimates $\boldsymbol{y}^0, \boldsymbol{u}^0 \in \mathbb{R}_+^K$
$\boldsymbol{x}^t = (\boldsymbol{\Omega}^t, \boldsymbol{w}^t, \boldsymbol{y}^t, \boldsymbol{u}^t)$
for $t = 0, 1, 2, \ldots$ **do**
  solve linear system (42) to get $\boldsymbol{\delta}^t$
  $\boldsymbol{x}^{t+1} = \boldsymbol{x}^t + \boldsymbol{\delta}^t$
  if

$$\begin{aligned} \left\| \begin{bmatrix} \boldsymbol{\gamma} - J^\intercal(\boldsymbol{\Omega}^t)\boldsymbol{y}^{t+1} \\ \nu - \boldsymbol{y}^{t+1} - \boldsymbol{u}^{t+1} \end{bmatrix} \right\| &\le \tau^{\frac{3}{2}} \\ ||(\boldsymbol{C}_{diag}(\boldsymbol{\Omega}^{t+1}) + \boldsymbol{W}^{\boldsymbol{t+1}}_{diag})\boldsymbol{y}^{t+1} - \tau|| &\le \tau \\ ||\boldsymbol{W}^{t+1}_{diag}\boldsymbol{u}^{t+1} - \tau|| &\le \tau \\ (\boldsymbol{y}^{t+1}, \boldsymbol{u}^{t+1}) &> \mathbf{0} \\ (\boldsymbol{c}(\boldsymbol{\Omega}^{t+1}) + \boldsymbol{w}^{t+1}, \boldsymbol{w}^{t+1}) &> \mathbf{0} \end{aligned} \tag{43}$$

  then
    $(\boldsymbol{\Omega}^{t+1}, \boldsymbol{w}^{t+1}, \boldsymbol{y}^{t+1}, \boldsymbol{u}^{t+1}) = \boldsymbol{x}^{t+1}$
    $\boldsymbol{\Omega}^* = \boldsymbol{\Omega}^{t+1}$
    return $\boldsymbol{\Omega}^*$
  else
    $\tau = \tau^{\frac{4}{3}}$

However, in Eq. (25), the instance number of microservices should be an integer, we should go back to $\mathbb{N}$ to find our optimal solutions. There are several methods helping us to solve this integer programming problem like decomposition method [30], cut-and-solve method [31] and branch-and-bound method [32]. In our work, we will choose the branch-and-bound method. The branch-and-bound algorithm enumerates candidate solutions with a rooted tree. By checking against upper and lower estimated bounds on the optimal solution, the algorithm traverses the rooted tree and terminates if it cannot produce a better solution than the best one. By branching the optimization problem with the bound of integer constraint in different steps, we can get the searching algorithm–the instance deployment algorithm for resource-constrained edges (*ID4ReE*), whose process is shown in Algorithm 3. It contains a branch-and-bound function (Algorithm 2) which finds the bounds and branches the searching space of the problem with a First-In-First-Out (FIFO) queue. The upper bound means the current minimum cost for solutions $\in \mathbb{N}^\theta$ while the lower bound stands for the minimum cost for solutions $\in \mathbb{R}_0^\theta$. A problem will be branched only when the minimizer is not integer and its corresponding cost is less than the upper bound (Algorithm 2 - Line 11-19).

5.1 Convergence Analysis

In this section, we will examine the convergence of the algorithms. Specifically, according to the structure of ID4ReE, we need to analyze both the interior point method and the branch-and-bound method. With an equivalent smooth reformulation of the penalty function, we can naturally adopt the primal-dual interior-point method to solve the problem. And with the help of [29] and [33], we have:

Algorithm 2. Branch and Bound, BnB

Input:
$\boldsymbol{\Omega}^\dagger$: the currently best solution;
lb, ub: the lower and upper bound of BnB;
$\boldsymbol{c}(\cdot)$: the constraint functions;
$\boldsymbol{\gamma}$: the cost vector;
ν: the penalty factor, $\nu > 0$
τ: the barrier factor, $\tau \in (0,1)$
Output:
$\boldsymbol{\Omega}^*$: the deployment scheme of instances;
1 Q = $Queue()$
2 Q.enqueue($\boldsymbol{c}$)
3 **while** Q *is not empty* **do**
4 $\boldsymbol{c}^o$ = Q.dequeue()
5 $\boldsymbol{\Omega}^o = IDA4ReE(\boldsymbol{\gamma}, \boldsymbol{c}^o, \tau, \nu)$
6 $v = \boldsymbol{\gamma}^T\boldsymbol{\Omega}^o$
7 **if** $\boldsymbol{\Omega}^o \in \mathbb{N}^\theta$ *and* $v \le ub$ **then**
8 $\boldsymbol{\Omega}^\dagger$, $ub = \boldsymbol{\Omega}^o$, v
9 **if** $\boldsymbol{\Omega}^o \notin \mathbb{N}^\theta$ **then**
10 **if** $v \le ub$ **then**
11 $lb = \min(lb, v)$
12 $k^* = \arg\max_{k \in [1,\theta], \boldsymbol{\Omega}^o_k \notin \mathbb{Z}} \gamma_k$
13 $Ik = \lfloor \boldsymbol{\Omega}^o_{k^*} \rfloor$
14 $\boldsymbol{u}_k = [\underbrace{0, \ldots, 0}_{k^*-1}, 1, \underbrace{0, \ldots, 0}_{\theta-k^*}]^\intercal$
15 $c_<(\boldsymbol{\Omega}) = Ik - \boldsymbol{u}_k^\intercal\boldsymbol{\Omega}$
16 $c_>(\boldsymbol{\Omega}) = \boldsymbol{u}_k^\intercal\boldsymbol{\Omega}$ - Ik - 1
17 $\boldsymbol{c}^l(\boldsymbol{\Omega}), \boldsymbol{c}^r(\boldsymbol{\Omega}) = \begin{bmatrix} \boldsymbol{c}(\boldsymbol{\Omega}) \\ c_<(\boldsymbol{\Omega}) \end{bmatrix}, \begin{bmatrix} \boldsymbol{c}(\boldsymbol{\Omega}) \\ c_>(\boldsymbol{\Omega}) \end{bmatrix}$
18 Q.enqueue($\boldsymbol{c}^l$)
19 Q.enqueue($\boldsymbol{c}^r$)
20 $\boldsymbol{\Omega}^* = \boldsymbol{\Omega}^\dagger$
21 **return** $\boldsymbol{\Omega}^*$

Lemma 1. *The stopping conditions are satisfied at* $\boldsymbol{x}^{t+1} = (\boldsymbol{x}_P^t, \boldsymbol{x}_D^t)$ *with* τ^t *for sufficiently large* t*, and*

$$||\Phi(\boldsymbol{x}_P^t, \boldsymbol{x}_D^t; \tau^t, \nu)|| = o(\tau^t). \tag{44}$$

Proof. The details are described in Theorem 6.2 of [29]. □

Lemma 2. *In Algorithm 1, the complete sequence* $\{\boldsymbol{x}^t\}$ *converges to* $\boldsymbol{x}^*$ *and sequence* $\{\Phi(\boldsymbol{x}^t; \tau^t, \nu)\}$ *converges to zero, the asymptotic convergence rate can be described with*

$$\frac{|\boldsymbol{x}^{t+2} - \boldsymbol{x}^*|}{|\boldsymbol{x}^{t+1} - \boldsymbol{x}^*|^{\frac{3}{2}}} = M, \tag{45}$$

where M *is a constant.*

Proof. It can be proved with Theorem 6.5 of [29] and Theorem 3.2 of [33] by setting $\{\gamma^k\}$ with $\{\frac{1}{2}\}$. □

TABLE 3
Configurations of Edge Service Provision Systems

ID #	m	n	μ qps	c MB	d MB	L_c MB	L_d GB	$\mathcal{B}$ MB/s	D^{in} MB	D^{out} MB	v_u MB/s	α	β
#1	2	5	[20, 50]	[100, 200]	[2, 5]	[512, 2048]	[32, 128]	[80, 100]	[1, 5]	[1, 5]	[1, 3]	10	25
#2	3	5	[20, 50]	[100, 200]	[2, 5]	[512, 2048]	[32, 128]	[80, 100]	[1, 5]	[10, 50]	[1, 3]	10	25
#3	4	5	[20, 50]	[100, 200]	[2, 5]	[512, 2048]	[32, 128]	[80, 100]	[1, 5]	[1, 5]	[1, 3]	10	25
#2	3	5	[20, 50]	[100, 200]	[2, 5]	[512, 2048]	[32, 128]	[80, 100]	[1, 5]	[1, 5]	[1, 3]	10	25
#4	3	3	[50, 60]	[100, 200]	[2, 5]	[512, 2048]	[32, 128]	[80, 100]	[1, 5]	[1, 5]	[1, 3]	10	25
#5	3	4	[50, 60]	[100, 200]	[2, 5]	[512, 2048]	[32, 128]	[80, 100]	[1, 5]	[1, 5]	[1, 3]	10	25
#6	3	5	[50, 60]	[100, 200]	[2, 5]	[512, 2048]	[32, 128]	[80, 100]	[1, 5]	[1, 5]	[1, 3]	10	25
#7	3	5	[20, 50]	[80, 100]	[2, 5]	[512, 2048]	[32, 128]	[80, 100]	[1, 5]	[1, 5]	[1, 3]	10	25
#8	3	5	[20, 50]	[100, 200]	[5, 6]	[512, 2048]	[32, 128]	[80, 100]	[1, 5]	[1, 5]	[1, 3]	10	25
#9	3	5	[20, 50]	[100, 200]	[2, 5]	[2048, 2560]	[32, 128]	[80, 100]	[1, 5]	[1, 5]	[1, 3]	10	25
#10	3	5	[20, 50]	[100, 200]	[2, 5]	[512, 2048]	[128, 160]	[80, 100]	[1, 5]	[1, 5]	[1, 3]	10	25
#11	3	5	[20, 50]	[100, 200]	[2, 5]	[512, 2048]	[32, 128]	[50, 60]	[1, 5]	[1, 5]	[1, 3]	10	25
#12	3	5	[20, 50]	[100, 200]	[2, 5]	[512, 2048]	[32, 128]	[80, 100]	[6, 7]	[1, 5]	[1, 3]	10	25
#13	3	5	[20, 50]	[100, 200]	[2, 5]	[512, 2048]	[32, 128]	[80, 100]	[1, 5]	[6, 7]	[1, 3]	10	25
#14	3	5	[20, 50]	[100, 200]	[2, 5]	[512, 2048]	[32, 128]	[80, 100]	[1, 5]	[1, 5]	[3, 4]	10	25
#15	3	5	[20, 50]	[100, 200]	[2, 5]	[512, 2048]	[32, 128]	[80, 100]	[1, 5]	[1, 5]	[1, 3]	20	25
#16	3	5	[20, 50]	[100, 200]	[2, 5]	[512, 2048]	[32, 128]	[80, 100]	[1, 5]	[1, 5]	[1, 3]	10	50

Lemma 1 shows the convergence, and by denoting the error after nth step in Lemma 2 with $\varepsilon_n = |\boldsymbol{x}^n - \boldsymbol{x}^*|$, we can iteratively estimate the error with

$$\lg \varepsilon_n + 2 \lg M = \tfrac{3}{2}(\lg \varepsilon_{n-1} + 2 \lg M). \tag{46}$$

Then when the precision of a d_p-dight number is required for IDA4ReE, the iteration number can be described with $O(\lg d_p)$ by solving Eq. (46). On the other hand, because the main process of the ID4ReE is the branch and bound algorithm, it will be extremely hard to determine when the integer solutions will occur. What's more, if no bounds are available in running this algorithm, the method will degenerate to an exhaustive search. To avoid this situation, we heuristically try to solve an integer linear programming (ILP) problem whose objective is $\max \sum_{i=1}^{\theta} \Omega_i$ and constraints are $c_{1\sim k}(\boldsymbol{\Omega}) \geq 0$. This is because the application is more likely to have smaller average response time if more microservice instances are deployed in the system. By selecting solutions that have $c_0(\boldsymbol{\Omega}) \geq 0$ from this ILP's solution set, we can roughly get the initial upper bound. Even though, the computation complexity may be as large as $O(\prod_{i=1,j=0}^{m,n+1} \min(\frac{L_c^j}{c_{i,j}}, \frac{L_d^j}{d_{i,j}}))$ in the worst case (here $\min(\frac{L_c^j}{c_{i,j}}, \frac{L_d^j}{d_{i,j}})$ determines the upper bound of $\Omega_{i,j}$ according to Eq. (22)).

6 Experiments and Analysis

6.1 Preliminary

We have implemented the proposed algorithms in Matlab 2018b and our experiments are conducted on a machine with Intel Xeon E5-2620 v4@2.10 GHz × 2 CPU and 64 GB memory on Windows 10 operating system. Due to the lack of well-adopted platforms and datasets, we generate a dataset for configurations of services and servers in a synthetic way for our experiment. Therefore, several edge service provisioning systems are created with the system configuration settings shown in Table 3. Though in many cases (e.g., like [24], [34] .etc) simulations are conducted on the single computer to check the results, here we try to use a multi-machine environment to make the results more convincing. Meanwhile, as we also want to investigate the factors that may affect the results by keeping other factors fixed, we finally turn to a powerful simulation tool whose name is *CloudSim* [35]. It can model the edge environments and measure the impact of resources, and many existing edge computing simulation platforms are built on it [36].

6.2 Baselines

Generally, researchers prefer to adopt some heuristic algorithms to solve the constrained nonlinear integer programming problem. Therefore, we choose some of the representative approaches as our baselines besides the brute-force (BF) one.

6.2.1 Genetic Algorithm

Genetic algorithm (GA) is one of the famous methods [34] which can be used for this purpose. GA simulates the evolution of populations with operations like selection, crossover, and mutation. It is designed to favor chromosomes with the highest fitness values to produce the next populations (solutions). As a result, the quality of solutions for a problem is gradually improved (population by population) until the optimal answer is reached.

1) *Selection*. Suppose the population is initialized with $C = (\boldsymbol{\Omega}^1, \boldsymbol{\Omega}^2, \ldots, \boldsymbol{\Omega}^P)$. Then in each successive generation, a portion of the existing population is selected

to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions are typically more likely to be selected. The fitness function measures the solution quality. In our approach, it can be defined by

$$F(\mathbf{\Omega}^i) = \frac{1}{\Psi(\mathbf{\Omega}^i, \nu)} \tag{48}$$

where ν is a large positive number. In this way, the kth solution will be selected with probability $P^k = \frac{F(\mathbf{\Omega}^k)}{\sum_{i=1}^{P} F(\mathbf{\Omega}^i)}$ to produce new generation.

2) *Crossover and Mutation*. To produce a new solution with crossover operation, a pair of "parent" chromosomes are selected with probability reflected in P^k. A new solution is created by exchanging parts of the selected parents with each other. On the other hand, the selected "parents" may choose not to crossover, then the new "offspring" are identical to themselves. The mutation operation changes some points of a solution, it gives the algorithm the ability to avoid premature convergence. At last, several solutions with good fitness will stay unchanged as elites in the next generation to keep the convergence. This process finally stops when converged after 5 consecutive iterations, it results in solutions with appropriate fitness values.

Algorithm 3. Instance Deployment Algorithm for Resource Constrained Edges, ID4ReE

Input:
$\boldsymbol{c}(\cdot)$: the constraint functions;
$\boldsymbol{\gamma}$: the cost vector;
ν: the penalty factor, $\nu > 0$
τ: the barrier factor, $\tau \in (0, 1)$
Output:
$\mathbf{\Omega}^*$: the deployment scheme of instances;

$\mathbf{\Omega}^\dagger, lb, ub = [\infty]_\theta^\intercal, \infty, \infty$
solve the following ILP, and get the solution set L:

$$\begin{aligned} \max \quad & \underbrace{[1, 1, \ldots, 1]}_{\theta}^{T} \cdot \mathbf{\Omega} \\ s.t. \quad & c_k(\mathbf{\Omega}) >= 0 (k = 1, \ldots, K), \mathbf{\Omega} \in \mathbb{Z}^\theta \end{aligned} \tag{47}$$

for $\mathbf{\Omega} \in L$ **do**
  if $c_0(\mathbf{\Omega}) \geq 0$ *and* $\boldsymbol{\gamma}^\intercal \mathbf{\Omega} < ub$ **then**
    $\mathbf{\Omega}^\dagger, lb, ub = \mathbf{\Omega}, \boldsymbol{\gamma}^\intercal \mathbf{\Omega}, \boldsymbol{\gamma}^\intercal \mathbf{\Omega}$
return $BnB(\mathbf{\Omega}^\dagger, lb, ub, \gamma, \nu, \tau)$

6.2.2 Teaching-Learning-Based Optimization Algorithm

The teaching-learning-based optimization (TLBO) algorithm was first proposed by Rao and Kalyankar [37]. TLBO is a population-based method that uses a population of solutions to proceed to the global solution. In TLBO, this population is named with "Class", in which "Teacher" is the optimal solution and "Learners" are the feasible solutions. When the ith feasible solution is denoted with $\mathbf{\Omega}^i$, the class C can be represented with $C = (\mathbf{\Omega}^1, \mathbf{\Omega}^2, \ldots, \mathbf{\Omega}^P)$. TLBO consists of two parts: "Teacher Phase" and "Student Phase".

1) *Teacher Phase*. The "Teacher Phase" means learning from the teacher. Every learner in the class will learn from the teacher through the difference between the teacher and the mean value of the learners

$$\begin{aligned} mean &= \frac{\sum_{i=1}^{P} \mathbf{\Omega}^i}{P} \\ diff &= r^i \times (\mathbf{\Omega}^{teacher} - TF^i \times mean) \\ \mathbf{\Omega}^i_{new} &= \mathbf{\Omega}^i_{old} + diff, \end{aligned} \tag{49}$$

here r^i = rand(0,1) is the learning step-length for $\mathbf{\Omega}^i$, and TF^i = round(1 + rand(0,1)) is the teaching factor for $\mathbf{\Omega}^i$. With the fitness function $F(\mathbf{\Omega}^i)$ in Eq. (48) as grade for $\mathbf{\Omega}^i$, all learners will update themselves with $\mathbf{\Omega}^i_{new}$ when $F(\mathbf{\Omega}^i_{new}) > F(\mathbf{\Omega}^i_{old})$.

2) *Student Phase*. The "Student Phase" means learning through interactions between learners. A learner learns something new if another learner has more knowledge, it can keep the population diverse. For each learner $\mathbf{\Omega}^i$ in the class, it will randomly choose a classmate $\mathbf{\Omega}^j$ ($i \neq j$) to see if it can learn something

$$\mathbf{\Omega}^i_{new} = \begin{cases} \mathbf{\Omega}^i_{old} + r^i(\mathbf{\Omega}^i - \mathbf{\Omega}^j), F(\mathbf{\Omega}^j) > F(\mathbf{\Omega}^i) \\ \mathbf{\Omega}^i_{old} + r^i(\mathbf{\Omega}^j - \mathbf{\Omega}^i), F(\mathbf{\Omega}^j) < F(\mathbf{\Omega}^i) \end{cases}. \tag{50}$$

6.2.3 Simulated Annealing Algorithm

Taking advantage of the idea from the annealing technique in metallurgy, the Simulated Annealing (SA) algorithm simulates the cooling steps for materials. It is a probabilistic technique for approximating the global optimum of a given function [38]. Suppose the current solution after the ith step is $\mathbf{\Omega}^i$ while the temperature is T^i. Then the energy of $\mathbf{\Omega}^i$ can be estimated with $E^i = F(\mathbf{\Omega}^i)$, where F is the fitness function shown in Eq. (48). To generate a new possible solution in step $i + 1$, SA will randomly select a solution $\mathbf{\Omega}^i_n$ from the neighborhood of $\mathbf{\Omega}^i$ in the searching space. Suppose E^i_n is the energy of solution $\mathbf{\Omega}^i_n$. Then we can compare E^i_n and E^i to produce the solution $\mathbf{\Omega}^{i+1}$

$$\mathbf{\Omega}^{i+1} = \begin{cases} \mathbf{\Omega}^i_n, E^i_n > E^i \quad or \quad P(E^i, E^i_n, T^i) > r^i \\ \mathbf{\Omega}^i, \ otherwise \end{cases}. \tag{51}$$

Here r^i = rand(0,1) is the acceptance threshold, and the acceptance probability $P(E^i, E^i_n, T^i)$ can be calculated with $P(E^i, E^i_n, T^i) = e^{-\frac{E^i - E^i_n}{kT^i}}$. After that, SA will conduct cooling operation $T^{i+1} = \mathcal{C} \cdot T^i$ to adjust the temperature ($\mathcal{C}$ is the cooling factor). With this policy, the algorithm will finally find some solutions with high energies.

6.3 Performance Comparison And Data Analysis

Based on the different configurations shown in Table 3, we construct 50 edge service provisioning systems for each of the configurations. For example, with configuration #1, we will construct a service provisioning system with 5 edge servers and the application to be deployed is made up of 2 microservices.

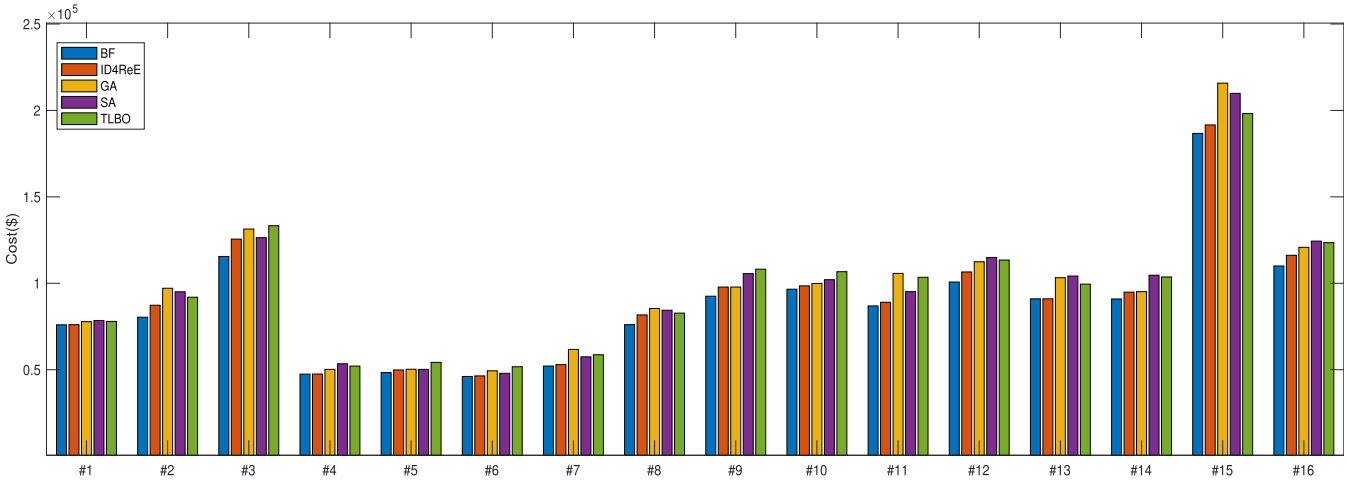

Fig. 6. The costs of deployment schemes generated by different approaches.

Besides these, we set the running parameters with $\mu_{i,j} \sim U(20, 50)$ qps, $c_{i,j} \sim U(100, 200)$ MB, $d_{i,j} \sim U(2, 5)$ GB, $L_c^j \sim U(512, 2048)$ GB, $L_j^d \sim U(32, 128)$ GB, $\mathcal{B}_{i,j} \sim U(80, 100)$ MB/s, $D_i^{in/out} \sim U(1, 5)$ MB, $v_u^j \sim U(1, 3)$ MB/s, and the price of computation/storage resource is set with α = 10$/MB and β = 25$/GB. Then we apply different approaches on these systems to find the appropriate deployment schemes. By evaluating the average cost, which is the objective of our problem for those systems with different configurations, we can explore how these factors will effect the performance of edge service provision system.

In Fig. 6, we illustrate the average costs of deployment schemes generated by different approaches on 50 edge service provisioning systems with a grouped bar chart. For each group of bars, they show the deployment costs for systems with some specific configurations. Namely, the costs of deployment schemes generated by GA, ID4ReE, SA and TLBO with the ith configuration set in Table 3. In summary, our approach performs better than other baselines, which means that it will cost less to deploy application microservices with our approach.

To go a step further, we then apply the deployment schemes generated by these approaches on those provisioning systems to explore the response time of requests. Therefore, we simulate 10,000 requests for every edge service provisioning system, and show the distributions of application response time for these requests for the 4 approaches with a heat-map. In the heat-map Fig. 7, the colored blocks stand for the distributions of the application response times. We can find that the application response times derived by ID4ReE is more concentrated, and most of application response times are less than 1.64s.

Besides the comparison between approaches, we will discuss what are the factors that may impact the results and how they will impact the results in the following parts.

6.3.1 Impact of Microservices

The number of microservices or the length of the service chain determines the complexity of an application. From Fig. 8 and the comparison of system #01, #02 and #03, we can find that the cost of the generated deployment scheme increases when the application becomes complex. Because more instances of the new microservices will be deployed to fulfill new tasks. Besides this, we can find that the cost increases when computation resource consumption or storage resource consumption of microservices becomes larger. This result is very clear because the cost is the linear weighting sum of the costs of computation resource consumption and storage resource consumption. At the same time, the cost increment caused by computation resource consumption is larger than that of storage resource consumption because of its larger unit price.

Microservice instances with larger processing capacity handle the requests more efficiently, which means the

Fig. 7. The distribution of application response time.

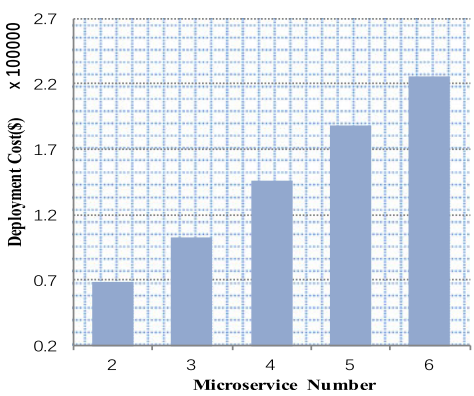

Fig. 8. Scheme costs for different microservice number.

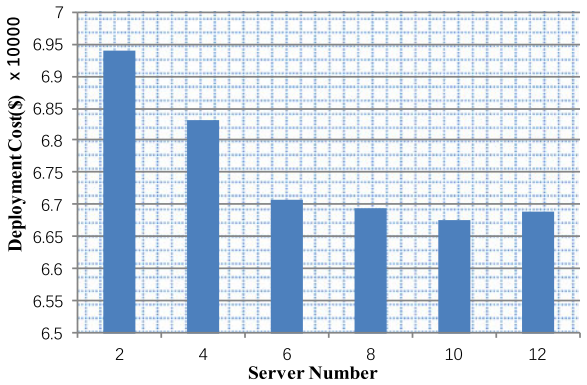

Fig. 9. Scheme costs for different server number.

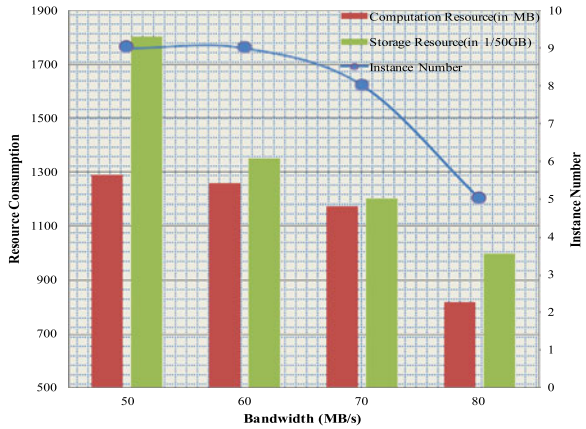

Fig. 10. The resource consumption and instance number under different bandwidth.

execution time can be reduced dramatically. Therefore, given the requirement of response time we will need fewer instances to fulfill the tasks of the application, as shown in Fig. 6 that the cost of #02 is less than that of #06. Besides these, the comparison of #12 and #13 points out that when the input and output size of microservices becomes smaller, the cost will also be less than before.

6.3.2 Impact of Pricing Policy

The pricing policy here means the price of different resources, it is determined by the infrastructure providers in general. Similar with the situation of microservice resource consumption, because the cost is proportional to the unit prices of computation resource and storage resource, we can find that a higher unit price of resources will cause the increasing of cost by comparing the results of #02 and #15, #16.

6.3.3 Impact of Servers

The edge server number, available resources and communication bandwidth determines the potential of the edge service provisioning system altogether if we regard the system as a distributed machine. The number of edge server determines the complexity of system topology. It provides more possibilities for deployment schemes. For example, if a new edge server es' which has the same parameters with es^* was added to the system, the instances deployed on es^* originally can be moved to es' partially without any loss while the risk resistance capacity can even increase. Not only comparing the results of #4, #5 and #6 in Fig. 6, we can go a step further with the result shown in Fig. 9. In this figure, we can find that the costs of these schemes decrease at first and then keep almost the same with the increasing of server numbers. This is caused by the changing of total available edge resources. Because there is no enough resource for the edge servers, microservice instances have to be deployed on the core server to offset the response time loss for the low transmission rate between edge servers and core server. Then, when the total available resource is enough so that most of the instances can be deployed on the edge servers, the generated deployment schemes will be similar.

The bandwidth is another important factor that can impact the deployment scheme. In reality, the transmission time is always a major part of response time. Therefore, we keep the configuration of a system fixed, and adjust the average bandwidth between servers and finally get Fig. 10 (to illustrate both the computation resource consumption and storage resource consumption, here we multiply the storage resource consumption with 50 in the figure). We can find that, by increasing the bandwidth between servers, the resource consumption and instance number both decrease. And these will directly decrease the cost of schemes. Besides the data transmission between servers, the data transmission between mobile devices and edge servers are also an important factor that can impact the results. By comparing the results of #2 and #14, it can be found that costs increase 8.32 percent when the transmission rate between the user and edge server becomes lower.

Besides the former relations, there is another interesting point that the microservice instances show a trend of aggregation in the deployment in the experiment. The aggregation here means that microservice instances would like to be deployed on the same server. For example, the list [2, 0, 2, 0] has a larger aggregation degree than the list [1, 1, 1, 1]. Naturally, here we use the variance to quantify the aggregation degree of instance number on server, and the results are shown in Fig. 11. In this figure we census the instance numbers on servers for different microservices in 50 edge service provision systems (the system configuration is #2), and illustrate the results with a scatter plot. In this figure, we can find that the variance becomes larger with the increasing of instances on the edge server, which means the aggregation degree becomes larger. This phenomenon can

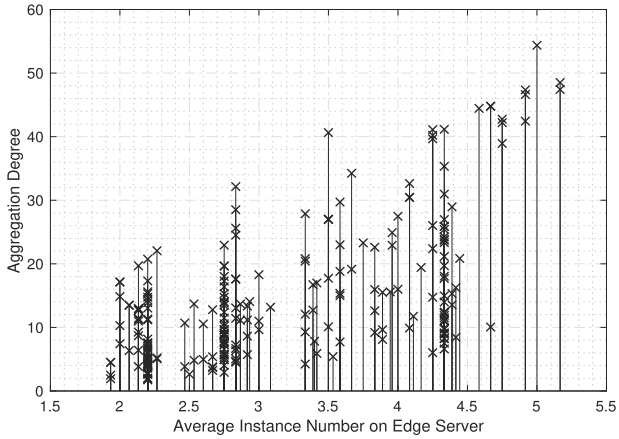

Fig. 11. The aggregation degree of microservice instances.

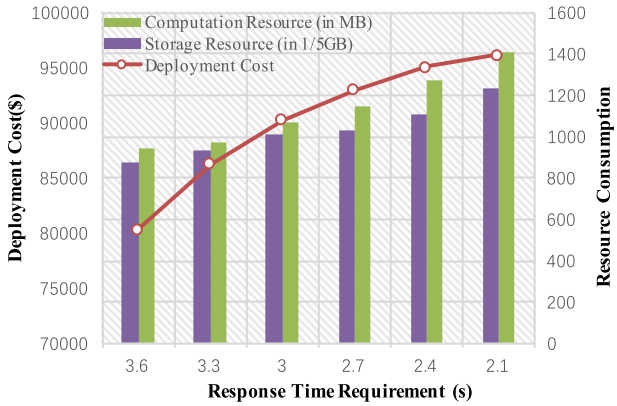

Fig. 12. The costs with different response requirement.

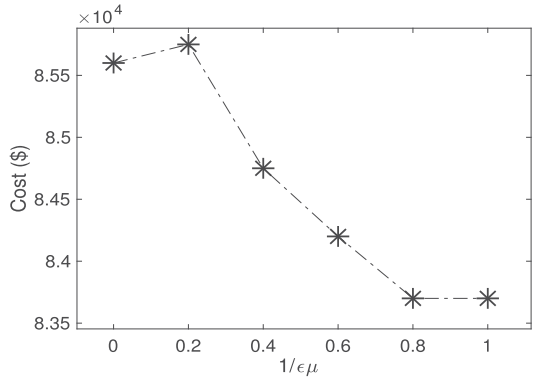

Fig. 13. The costs with different ϵ.

be explained with the queuing theory, because that the sojourn time of M/M/c system is less than c parallel M/M/1 systems.

6.3.4 Impact of Response Requirement

The requirement of response time reflects the developers' expectations for their application. In many cases, the application providers need a trade-off between performance and cost, so they should be more careful about the balance so that they can save money as well as keep the quality of experience. In Fig. 12, we draw the curve for deployment schemes with different application response time requirements to show this relation (to illustrate both the computation resource consumption and storage resource consumption, here we multiply the storage resource consumption with 5 in the figure). We can find that there is an obvious trend that the cost increases when we want to have a lower response time. By drawing figures for different systems, the application developers will get a general idea of the cost they have to pay for given response requirements. This will help the developers balance the cost and performance.

6.3.5 Impact of Queue Time Approximation

As we use the approximation form $\mathbb{E}[\hat{T}(\mathbf{\Omega})]$ instead of $\mathbb{E}[T(\mathbf{\Omega})]$ to simplify the problem in this work, some extra loss in accuracy maybe introduced for it. Though the gap between $\mathbb{E}[\hat{T}(\mathbf{\Omega})]$ and $\mathbb{E}[T(\mathbf{\Omega})]$ can be estimated with Eqs. (29) and (31), we also need to check how it will affect the results with experiments. Therefore, we choose $\epsilon = [\frac{1}{5\mu_{min}}, \frac{2}{5\mu_{min}}, \frac{3}{5\mu_{min}}, \frac{4}{5\mu_{min}}, \frac{1}{\mu_{min}}]$ and Eq. (31) to find service deployment scheme individually, and compare them with response time constraint Eq. (21) and $\epsilon = 0$. The comparison result is shown in Fig. 13. In this figure, we can find that: compared with the original form, the deployment scheme generated with the approximation form can get more or less cost with different choice of ϵ. This is easy to understand because the real gap ϵ_r may locate in $[0, \epsilon)$ or $[\epsilon, \frac{1}{\mu_{min}}]$: a smaller guess for ϵ_r may result in a smaller response time requirement, and the corresponding deployment scheme will result in a smaller average response time but a higher cost; a larger guess for ϵ_r may result in a larger response time requirement, and the corresponding deployment scheme will result in a larger average response time but a lower cost.

6.3.6 A Guide for Service Deployment

To go a step further, we can investigate the relationship between the instance number of microservice and the states of different edge servers to explore the best practice of deploying. In this way, the first thing to do is to represent the state of an edge server. As the edge server will provide resources, process requests and communicate with different devices, we can describe its state from these perspectives. From the perspective of resource provision, every edge server can provide computation and storage resources, which are represented with L_c and L_d. From the perspective of communication efficiency, the communication of every edge server can be classified into machine-to-machine (M2M) communication and machine-to-device (M2D) communication. The efficiency of M2M can be defined with

$$e_{M2M} = \frac{1}{n} \sum_{s' \in ES, s' \neq s} \frac{1}{\mathcal{B}_{s,s'}}. \tag{52}$$

And the efficiency of M2D can be defined with

$$e_{M2D} = \begin{cases} \frac{1}{v_u^s} & , s \in ES \\ \frac{1}{n}\sum_{s=1}^{n} \frac{1}{v_u^s} + \frac{1}{n}\sum_{s' \in ES} \frac{1}{\mathcal{B}_{s,s'}} & , s \in CS \end{cases}, \tag{53}$$

these two definitions measure the transmission efficiency for 1 MB data to users and servers. From the perspective of request processing, the efficiency can be described by the average processing capacity $\hat{\mu}_j = \frac{\Omega_{i,j}\mu_{i,j}}{\sum_{i=1}^{m}\Omega_{i,j}}$ and active request to an edge server, which can be quantified with λ_j. Therefore, we can represent the state of an edge server with these 6 indicators.

In general, Pearson correlation coefficient can be used to describe the relation between two sequence X and Y.

$$r_p(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}. \tag{54}$$

Therefore, by using Pearson correlation coefficient to calculate the correlation between the instance number and these indicators, we can get the radar plot shown in Fig. 14.

In this figure, we can find that the deployed instance number of an edge server is related to the processing capacity and the requests generated from the devices in its serving area. This will be reasonable because deploying instances on these edge servers will result in a reduction in execution and

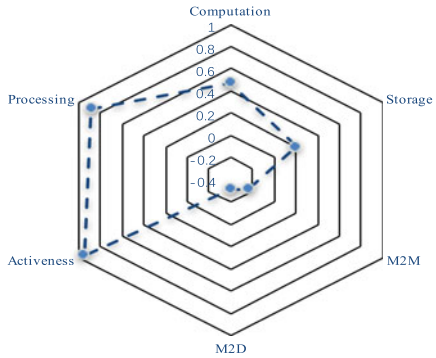

Fig. 14. The correlations between instance number and 6 dimensions for microservices.

transmission. Compared with the storage resource, the computation resource will affect more on the deployment schemes. This is true according to our settings, because the computation resource will be the bottleneck in most cases. Therefore, a heuristic guide for the developers is to put microservice instances on the servers whose processing capacity is better and whose users in the serving area are more active.

7 Related Work

With an increasing number of mobile devices connecting to the cloud, the demand for high-quality service provision becomes urgent. It drives more and more researchers to pay attention to issues of the MEC model that affects the effectiveness of service provision. In this section, we review the research related to our study, i.e., the research about MEC framework and service deployment.

7.1 MEC Framework

With the help of a MEC model, researchers and developers reconstruct their system components to achieve their different goals. Since the MEC model focuses on mobile end devices, energy consumption reduction and performance optimization become the main research topics to perform computation in an economical and efficient way. For example, Li *et al.* [39] consider the energy consumption of mobile devices; they analyze overheads of mobile devices and propose an overhead-optimizing multi-device task scheduling strategy for ad-hoc-based MEC systems. Stefania *et al.* [40] consider a Multi-Input and Multi-Output (MIMO) multi-cell system where multiple mobile users ask for computation offloading to servers; they formulate the offloading problem as the joint optimization of the radio resources and the computational resources to satisfy the latency constraints. Yu *et al.* [41] consider incorporation with dense cellular networks; they propose an online algorithm based on Lyapunov optimization to determine offloading and edge server sleeping policy and increase performance while keeping low energy consumption. Yi *et al.* [42] propose LAVEA, a system built for edge computing, which offloads computation tasks between clients and edge nodes, collaborates nearby edge nodes, to provide low-latency video analytics at places closer to the users.

7.2 Microservice Deployment

The nature of microservice deployment problems is an assignment problem. By considering different types of constraints in reality, researchers have done lots of work on it. Fan *et al.* [6] consider the application workloads among cloudlet and propose an application-aware workload allocation scheme for edge to minimize the response time of application requests by assigning requests to appropriate destination cloudlets. And they also consider the workload balance problem [43], they propose a workload balancing scheme to minimize the latency of data flows by associating devices to suitable edge servers. Huang *et al.* [44] present a load-aware service deployment approach for dynamic workloads and a service request scheduling method based on task ranking mechanisms to improve the execution performance of composite cloud services in dynamic cloud environments. Moens *et al.* [45] present and evaluate a formal model for resource allocation of virtualized network functions within NFV environments. Wu *et al.* [46] have proposed an elastic framework to automatically and dynamically deploy cloud services on data center, base stations, client units, even peer devices, so that all available resources around mobile users are made use of to provide seamless service. Li *et al.* [47] study the joint problem of service function chain deploying and path selection for bandwidth saving and virtual network functions reuse, model it as a multi-objective problem and propose a heuristic approach to solve it. Wu *et al.* [46] propose an elastic framework that can automatically and dynamically deploy cloud services on data center, base stations, client units, even peer devices based on their context-aware model so that the cost can be optimized. Vögler *et al.* [10] present a framework for the dynamic generation of optimized deployment topologies for IoT cloud applications that are tailored to the currently available physical infrastructure so that the application components on edge devices can provide service flexibly.

These researches shed light on the fundamental concepts involved in the cache problem in MEC models. Based on their work, we can go a step further to explore how to deploy microservice instances on the mobile edge servers for the microservice-based applications with both performance and cost requirements.

8 Conclusion and Future Work

This paper introduces the mobile edge computing model and highlights the scenario of deploying the microservice-based application on the edge service provisioning system. Based on them, we model the microservice instances on servers as the queuing node and propose an approach to find optimal deployment schemes with lower cost while meeting the demand of application average response time. In addition, we explore the factors that may affect the results and give some guidance to developers in deploying microservice-based applications.

As the approach can generate deployment schemes for applications when the location-aware request arrival rate λ is given in a time period, the application developers can dynamically update the deployment schemes when the request arrival rate can be predicted accurately. It means that we can turn to develop some prediction models in future work. Besides this, the routing policy can also act as a decision variable. If we can determine the routing policy as well as the deployment scheme, there also may be an improvement in saving the cost.

ACKNOWLEDGMENTS

This work was supported in part by the National Key Research and Development Program of China (No. 2017 YFB1400601), National Science Foundation of China (No. 617 72461 & No. 61825205) and Natural Science Foundation of Zhejiang Province (No. LR18F020003).

REFERENCES

[1] A. Paya and D. C. Marinescu, "Energy-aware load balancing and application scaling for the cloud ecosystem," *IEEE Trans. Cloud Comput.*, vol. 5, no. 1, pp. 15–27, Jan.–Mar. 2017.

[2] K. Li, "Improving multicore server performance and reducing energy consumption by workload dependent dynamic power management," *IEEE Trans. Cloud Comput.*, vol. 4, no. 2, pp. 122–137, Apr.–Jun. 2016.

[3] S. Deng, H. Wu, W. Tan, Z. Xiang, and Z. Wu, "Mobile service selection for composition: An energy consumption perspective," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 3, pp. 1478–1490, Jul. 2017.

[4] M. Patel *et al.*, "Mobile-edge computing introductory technical white paper," 2014, Accessed: Jul. 4, 2018. [Online]. Available: https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1\%2018-09-14.pdf

[5] S. Wang, Y. Zhao, L. Huang, J. Xu, and C.-H. Hsu, "QoS prediction for service recommendations in mobile edge computing," *J. Parallel Distrib. Comput.*, vol. 127, pp. 134–144, 2017.

[6] Q. Fan and N. Ansari, "Application aware workload allocation for edge computing-based IoT," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 2146–2153, Jun. 2018.

[7] I. Filip, F. Pop, C. Serbanescu, and C. Choi, "Microservices scheduling model over heterogeneous cloud-edge environments as support for IoT applications," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2672–2681, Aug. 2018.

[8] P. D. Francesco, P. Lago, and I. Malavolta, "Migrating towards microservice architectures: An industrial survey," in *Proc. IEEE Int. Conf. Softw. Architecture*, 2018, pp. 29–39.

[9] F. Boyer, X. Etchevers, N. D. Palma, and X. Tao, "Architecture-based automated updates of distributed microservices," in *Proc. 16th Int. Conf. Service-Oriented Comput.*, 2018, pp. 21–36.

[10] M. Vögler, J. M. Schleicher, C. Inzinger, and S. Dustdar, "Optimizing elastic IoT application deployments," *IEEE Trans. Services Comput.*, vol. 11, no. 5, pp. 879–892, Sep./Oct. 2018.

[11] S. Nastic, H. L. Truong, and S. Dustdar, "Data and control points: A programming model for resource-constrained IoT cloud edge devices," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, 2017, pp. 3535–3540.

[12] H. Xu *et al.*, "Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in web applications," in *Proc. Web Conf.*, 2018, pp. 187–196.

[13] S. Deng, Z. Xiang, J. Yin, J. Taheri, and A. Y. Zomaya, "Composition-driven IoT service provisioning in distributed edges," *IEEE Access*, vol. 6, pp. 54 258–54 269, 2018.

[14] M. D. Cia *et al.*, "Using smart city data in 5G self-organizing networks," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 645–654, Apr. 2018.

[15] S. Dustdar, S. Nastic, and O. Scekic, *Smart Cities - The Internet of Things, People and Systems*. Berlin, Germany: Springer, 2017.

[16] M. Vögler, J. M. Schleicher, C. Inzinger, S. Dustdar, and R. Ranjan, "Migrating smart city applications to the cloud," *IEEE Cloud Comput.*, vol. 3, no. 2, pp. 72–79, Mar./Apr. 2016.

[17] A. Gamez-Diaz, P. Fernandez, and A. Ruiz-Cortes, "An analysis of restful APIs offerings in the industry," in *Proc. Int. Conf. Service-Oriented Comput.*, 2017, pp. 589–604.

[18] C. Ding, A. Zhou, J. Huang, Y. Liu, and S. Wang, "ECDU: An edge content delivery and update framework in mobile edge computing," *EURASIP J. Wireless Commun. Netw.*, vol. 2019, no. 1, 2019, Art. no. 268.

[19] P. Burke, "The output process of a stationary M/M/s queueing system," *The Ann. Math. Statist.*, vol. 39, no. 4, pp. 1144–1152, 1968.

[20] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.

[21] W.-P. Yang, L.-C. Wang, and H.-P. Wen, "A queueing analytical model for service mashup in mobile cloud computing," in *Proc. IEEE Wireless Commun. Netw. Conf.*, 2013, pp. 2096–2101.

[22] Y. Xiao, C. Lin, Y. Jiang, X. Chu, and X. Shen, "Reputation-based QoS provisioning in cloud computing via dirichlet multinomial model," in *Proc. IEEE Int. Conf. Commun.*, 2010, pp. 1–5.

[23] D. Chouhan, S. D. Kumar, and J. A. Ajay, "A MLFQ scheduling technique using M/M/c queues for grid computing," *Int. J. Comput. Sci. Issues*, vol. 10, no. 2 Part 1, 2013, Art. no. 357.

[24] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 207–215.

[25] Z. Xiang, S. Deng, J. Taheri, and A. Zomaya, "Dynamical service deployment and replacement in resource-constrained edges," *Mobile Netw. Appl.*, pp. 1–16, 2019. [Online]. Available: https://link.springer.com/article/10.1007/s11036-019-01449-7

[26] T. Antczak, "The L1 penalty function method for nonconvex differentiable optimization problems with inequality constraints," *Asia-Pacific J. Oper. Res.*, vol. 27, no. 05, pp. 559–576, 2010.

[27] A. Forsgren and P. E. Gill, "Primal-dual interior methods for nonconvex nonlinear programming," *SIAM J. Optim.*, vol. 8, no. 4, pp. 1132–1152, 1998.

[28] A. R. Conn, N. I. Gould, D. Orban, and P. L. Toint, "A primal-dual trust-region algorithm for non-convex nonlinear programming," *Math. Program.*, vol. 87, no. 2, pp. 215–249, 2000.

[29] N. I. Gould, D. Orban, A. Sartenaer, and P. L. Toint, "Superlinear convergence of primal-dual interior point algorithms for nonlinear programming," *SIAM J. Optim.*, vol. 11, no. 4, pp. 974–1002, 2001.

[30] F. Yang, K. Gao, I. W. Simon, Y. Zhu, and R. Su, "Decomposition methods for manufacturing system scheduling: A survey," *IEEE/CAA J. Automatica Sinica*, vol. 5, no. 2, pp. 389–400, Mar. 2018.

[31] P. Wu, A. Che, F. Chu, and M. Zhou, "An improved exact ε-constraint and cut-and-solve combined method for biobjective robust lane reservation," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 3, pp. 1479–1492, Jun. 2015.

[32] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell, "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning," *Discr. Optim.*, vol. 19, pp. 79–102, 2016.

[33] N. I. Gould, D. Orban, A. Sartenaer, and P. L. Toint, "Componentwise fast convergence in the solution of full-rank systems of nonlinear equations," *Math. Program.*, vol. 92, no. 3, pp. 481–508, 2002.

[34] S. Deng, L. Huang, J. Taheri, J. Yin, M. Zhou, and A. Y. Zomaya, "Mobility-aware service composition in mobile communities," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 3, pp. 555–568, Mar. 2017.

[35] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *Proc. Int. Conf. High Perform. Comput. Simul.*, 2009, pp. 1–11.

[36] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments," *Softw. Pract. Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[37] R. V. Rao, V. J. Savsani, and D. Vakharia, "Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems," *Comput.-Aided Des.*, vol. 43, no. 3, pp. 303–315, 2011.

[38] S. Lyden and M. E. Haque, "A simulated annealing global maximum power point tracking approach for PV modules under partial shading conditions," *IEEE Trans. Power Electron.*, vol. 31, no. 6, pp. 4171–4181, Jun. 2016.

[39] L. Tianze, W. Muqing, Z. Min, and L. Wenxing, "An overhead-optimizing task scheduling strategy for ad-hoc based mobile edge computing," *IEEE Access*, vol. 5, pp. 5609–5622, 2017.

[40] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 1, no. 2, pp. 89–103, Jun. 2015.

[41] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.

[42] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "LAVEA: Latency-aware video analytics on edge computing platform," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 2573–2574.
[43] Q. Fan and N. Ansari, "Towards workload balancing in fog computing empowered IoT," *IEEE Trans. Netw. Sci. Eng.*, to be published, doi: 10.1109/TNSE.2018.2852762.
[44] K.-C. Huang, Y.-C. Lu, M.-H. Tsai, Y.-J. Wu, and H.-Y. Chang, "Performance-efficient service deployment and scheduling methods for composite cloud services," in *Proc. 9th Int. Conf. Utility Cloud Comput.*, 2016, pp. 240–244.
[45] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *Proc. 10th Int. Conf. Netw. Service Manage. Workshop*, 2014, pp. 418–423.
[46] K. Wu, W. Liu, and S. Wu, "Dynamic deployment and cost-sensitive provisioning for elastic mobile cloud services," *IEEE Trans. Mobile Comput.*, vol. 17, no. 6, pp. 1326–1338, Jun. 2018.
[47] D. Li, J. Lan, and P. Wang, "Joint service function chain deploying and path selection for bandwidth saving and VNF reuse," *Int. J. Commun. Syst.*, vol. 31, no. 6, 2018, Art. no. e3523.

Shuiguang Deng (Senior Member, IEEE) received the BS and PhD degrees both in computer science from Zhejiang University, China, in 2002 and 2007, respectively. He is currently a full professor with the College of Computer Science and Technology, Zhejiang University. He was previously a visiting scholar worked with the Massachusetts Institute of Technology, Cambridge, Massachusetts, in 2014, and at Stanford University, Stanford, California, in 2015 as a visiting scholar. His research interests include edge computing, service computing, mobile computing, and business process management. He serves as associate editor for the journal *IEEE Access* and IET Cyber-Physical Systems: Theory & Applications as an associate editor. During the past ten years, he has published more than 90 papers in journals, such as the *IEEE Transactions on Computers*, *IEEE Transactions on Parallel & Distributed Systems*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Cybernetics* and *IEEE Transactions on Neural Networks and Learning Systems*, and refereed conferences including WWW, ER, ICWS, ICSOC, *etc.* In 2018, he was granted the Rising Star Award by IEEE TCSVC.

Zhengzhe Xiang (Student Member, IEEE) received the bachelor's degree of computer science and technology in Zhejiang University, China. He is currently working toward the PhD degree in the College of Computer Science, Zhejiang University, China. His research interests lie in the fields of service computing, cloud computing, and edge computing.

Javid Taheri (Member, IEEE) received the bachelor's and master's degrees in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 1998 and 2000, respectively, and the PhD degree in mobile computing from the School of Information Technologies, University of Sydney, Australia. Since 2006, he has been actively working in several fields, including: Networking, optimization, parallel/distributed computing, and cloud computing. Because of for his contribution to the vibrant area of cloud computing, he was selected among 200 top young rehearses by the Heidelberg Forum, in 2013. He also holds several cloud/networking related industrial certification from VMware (VCP-DCV, VCP-DT, and VCP-Cloud), Cisco (CCNA), Microsoft, etc. He is currently working as an associate professor at the Department of Computer Science, Karlstad University, Sweden. His major areas of interest include (1) profiling, modeling and optimization techniques for private and public cloud infrastructures, (2) profiling, modeling and optimization techniques for software defined networks, and (3) network-aware scheduling algorithms for cloud and green computing.

Mohammad Ali Khoshkholghi received the bachelor's and master's degrees of computer engineering, in 2007 and 2010, respectively, and the PhD degree in computer science from the Faculty of Computer Science and Information Technology, University Putra Malaysia, Seri Kembangan, Malaysia, in 2017. He is currently a postdoctoral research fellow at the Department of Computer Science, Karlstad University, Sweden. Before joining KAU, he worked as researcher and university lecturer within computer science. His research interests lie in the area of edge and cloud computing, network function virtualization and optimization techniques.

Jianwei Yin (Member, IEEE) received the PhD degree from Zhejiang University, China, in 2001. He is a full professor with the College of Computer Science and Technology, Zhejiang University, China. His current research interests include cloud computing, performance evaluation, service computing, middleware, etc. He has published more than 120 research papers in major peer-reviewed international journals and conference proceedings in these areas. He is the associate editor of the *Transaction on Service Computing*.

Albert Y. Zomaya (Fellow, IEEE) is currently a chair professor of high performance computing & networking in the School of Information Technologies, The University of Sydney, Sydney, Australia. He is also the director with the Centre for Distributed and High Performance Computing, which was established in late 2009. He has published more than 500 scientific papers and articles, and is a author, co-author or editor of more than 20 books. He served as the editor-in-chief of the *IEEE Transactions on Computers* (2011-2014). He serves as an associate editor for 22 leading journals, such as the *ACM Computing Surveys*, *IEEE Transactions on Computational Social Systems*, *IEEE Transactions on Cloud Computing*, and *Journal of Parallel and Distributed Computing*. He delivered more than 150 keynote addresses, invited seminars, and media briefings and has been actively involved, in a variety of capacities, in the organization of more than 600 national and international conferences. He received the IEEE Technical Committee on Parallel Processing Outstanding Service Award (2011), IEEE Technical Committee on Scalable Computing Medal for Excellence in Scalable Computing (2011), and IEEE Computer Society Technical Achievement Award (2014). He is a chartered engineer, a fellow of AAAS, and IET (U. K.). His research interests include the areas of parallel and distributed computing and complex systems.

Schahram Dustdar (Fellow, IEEE) is currently a full professor of computer science (informatics) with a focus on Internet Technologies heading the Distributed Systems Group at the TU Wien. He is the chairman of the Informatics Section of the Academia Europaea (since December 9, 2016). He is elevated to IEEE fellow (since January 2016). From 2004-2010 he was honorary professor of Information Systems at the Department of Computing Science, University of Groningen (RuG), The Netherlands. From December 2016 until January 2017, he was a visiting professor with the University of Sevilla, Spain, and from January until June 2017, he was a visiting professor at UC Berkeley, Berkeley, California. He is a member of the IEEE Conference Activities Committee (CAC) (since 2016), of the Section Committee of Informatics of the Academia Europaea (since 2015), a member of the Academia Europaea: The Academy of Europe, Informatics Section (since 2013). He is recipient of the ACM distinguished scientist award (2009) and IBM Faculty Award (2012). He is an associate editor of the *IEEE Transactions on Services Computing*, *ACM Transactions on the Web*, and *ACM Transactions on Internet Technology* and on the editorial board of the *IEEE Internet Computing*. He is the editor-in-chief of the *Computing* (an SCI-ranked journal of Springer).