

# Distributed Redundant Placement for Microservice-based Applications at the Edge

Hailiang Zhao, Shuiguang Deng, *Senior Member, IEEE*, Zijie Liu, Jianwei Yin,  
and Schahram Dustdar, *Fellow, IEEE*

**Abstract**—Multi-access Edge Computing (MEC) is booming as a promising paradigm to push the computation and communication resources from cloud to the network edge to provide services and to perform computations. With container technologies, mobile devices with small memory footprint can run composite microservice-based applications without time-consuming backbone. Service placement at the edge is of importance to put MEC from theory into practice. However, current state-of-the-art research does not sufficiently take the composite property of services into consideration. Besides, although Kubernetes has certain abilities to heal container failures, high availability cannot be ensured due to heterogeneity and variability of edge sites. To deal with these problems, we propose a distributed redundant placement framework SAA-RP and a GA-based Server Selection (GASS) algorithm for microservice-based applications with sequential combinatorial structure. We formulate a stochastic optimization problem with the uncertainty of microservice request considered, and then decide for each microservice, how it should be deployed and with how many instances as well as on which edge sites to place them. Benchmark policies are implemented in two scenarios, where redundancy is allowed and not, respectively. Numerical results based on a real-world dataset verify that GASS significantly outperforms all the benchmark policies.

**Index Terms**—Redundancy, Service Placement, Multi-access Edge Computing, Composite Service, Sample Average Approximation.

## 1 INTRODUCTION

NOWADAYS, mobile applications are becoming more and more computation-intensive, location-aware, and delay-sensitive, which puts a great pressure on the traditional Cloud Computing paradigm to guarantee the Quality of Service (QoS). To address the challenge, Multi-access Edge Computing (MEC) was proposed to provide services and to perform computations at the network edge without time-consuming backbone transmission, so as to enable fast responses for mobile devices [1] [2] [3].

MEC offers not only the development on the network architecture, but also the innovation in service patterns. Considering that small-scale data-centers can be deployed near cellular tower sites, there are exciting possibilities that microservice-based applications can be delivered to mobile devices without backbone transmission, in virtue of setting up a unified service provision platform. Container technologies, represented by Docker [4], and its dominant orchestration and maintenance tool, Kubernetes [5], are becoming the mainstream solution for packaging, deploying, maintaining, and healing applications. Each microservice decoupled from the application can be packaged as a Docker image and each microservice instance is a Docker container. Here we take Kubernetes for example. Kubernetes is naturally suitable for building *cloud-native* applications by leveraging the benefits of the distributed edge because it can hide the complexity of microservice orchestration while managing their availability

with lightweight Virtual Machines (VMs), which greatly motivates Application Service Providers (ASPs) to participate in service provision within the access and core networks.

Service deployment from ASPs is the carrier of service provision, which touches on where to place the services and how to deploy their instances. In the last two years, there exist works study the placement at the network edge from the perspective of Quality of Experience (QoE) of end users or the budget of ASPs [6] [7] [8] [9] [10] [11] [12]. However, those works commonly have two limitations. Firstly, the *to-be-deployed* service only be studied in an atomic way. It is often treated as a single abstract function with given input and output data size. Time series or composition property of services are not fully taken into consideration. Secondly, high availability of deployed service is not carefully studied. Due to the heterogeneity of edge sites, such as different CPU cycle frequency and memory footprint, varying background load, transient network interrupts and so on, the service provision platform might face greatly slowdowns or even runtime crash. However, the default assignment, deployment, and management of containers does not *fully* take the heterogeneity in both physical and virtualized nodes into consideration. Besides, the healing capability of Kubernetes is principally monitoring the status of containers, pods, and nodes and timely restarting the failures, which is not enough for high availability. Vayghan et al. find that in the specific test environment, when the pod failure happens, the outage time of the corresponding service could be dozens of seconds. When node failure happens, the outage time could be dozens of minutes [13] [14]. Therefore, with the vanilla version of Kubernetes, high availability might not be ensured, especially for the latency-critical cloud-native applications. Besides, one microservice could have several

- H. Zhao, S. Deng, Z. Liu, and J. Yin are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310058, China. E-mail: {hliangzhao, dengsg, liuzijie, zjuyjw}@zju.edu.cn
- S. Dustdar is with the Distributed Systems Group, Technische Universitt Wien, 1040 Vienna, Austria. E-mail: dustdar@dsg.tuwien.ac.at
- Shuiguang Deng is the corresponding author.

alternative execution solutions. For example, electronic payment, as a microservice of a composite service, can be executed by PayPal, WeChat Pay, and AliPay<sup>1</sup>. In this paper, let us call them *candidates (of microservices)*. This status quo complicates the placement problem further. Because it is the instances of candidates that need to be placed, which greatly scales up the problem.

In order to solve the above problems, we propose a distributed redundant placement framework, i.e., Sample Average Approximation-based Redundancy Placement (SAA-RP), for a microservice-based applications with sequential combinatorial structure. For this application, if all of the candidates are placed on one edge site, network congestion is inevitable. Therefore, we adopt a distributed placement scheme, which is naturally suitable for the distributed edge. Redundancy is the core of SAA-RP, which allows that one candidate to be dispatched to multiple edge sites. By creating multiple candidate instances, it boosts a faster response to service requests. To be specific, it alleviates the risk of a long delay incurred when a candidate is assigned to only one edge site. With one candidate deployed on more than one edge site, requests from different end users at different locations can be balanced, so as to ensure the high availability of service and the robustness of the provision platform. Actually, performance of redundancy has been extensively studied under various system models and assumptions, such as the Redundancy- $d$  model, the  $(n, k)$  system, and the  $S&X$  model [15]. However, which kind of candidate requires redundancy and how many instances should be deployed cannot be decided if out of a concrete situation. Currently, the main strategy of job redundancy usually releases the resource occupancy after completion, which is not befitting for geographically distributed edge sites. This is because service requests are continuously generated from different end users. The destruction of candidate instances have to be created again, which will certainly lead to the delay in service responses. Besides, redundancy is not always a win and might be dangerous sometimes, since practical studies have shown that creating too many instances can lead to unacceptably high response times and even instability [15].

As a result, we do not release the candidate instances but periodically update them based on the observations of service demand status during that period. Specifically, we derive expressions to decide each candidate should be dispatched with how many instances and which edge sites to place them. By collecting user requests for different service composition schemes, we model the distributed redundant placement as a stochastic discrete optimization problem and approximate the expected value through Monte Carlo sampling. During each sampling, we solve the deterministic problem based on an efficient evolutionary algorithm. Performance analysis and numerical results are provided to verify its practicability. Our main contributions are as follows.

- 1) We model the distributed placement scenario at the edge for general microservice-based chained applications and design a distributed redundant place-

1. Both Alipay and WeChat Pay are third-party mobile and online payment platforms, established in China.

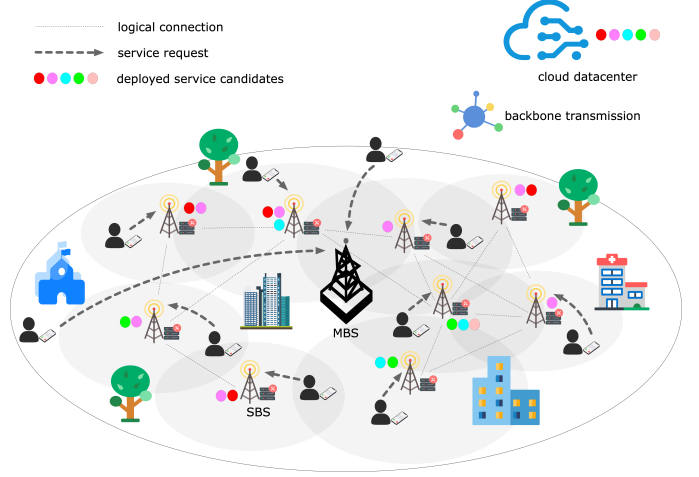


Fig. 1. A typical scenario for a pre-5G HetNet.

ment framework SAA-RP. SAA-RP can decide each candidate should be dispatched with how many instances and which edge sites to placement them. It makes up with the shortcoming of the default scheduler of Kubernetes, i.e. `kube-scheduler` [16], when encountering the MEC.

- 2) We take both the uncertainty of end users' service requests and the heterogeneity of edge sites into consideration and formulate a stochastic optimization problem. Based on the long-term observation on end users' service requests, we approximate the stochastic problem by sampling and averaging.
- 3) Simulations are conducted based on the real-world EUA Dataset [17]. We also provide the performance analysis on algorithm optimality and convergence rate. The numerical results verify the practicability and superiority of our algorithm, compared with several typical benchmark policies.

The organization of this paper is as follows. Section 2 demonstrates the motivation scenario. Section 3 introduces the system model and formulates a stochastic optimization problem. The SAA-RP framework is proposed in Section 4 and its performance analysis is conducted in Section 5. We show the simulation results in Section 6. In Section 7, we review related works on service placement at the edge and typical redundancy models. Section 8 concludes this paper.

## 2 MOTIVATION SCENARIOS

### 2.1 The Heterogeneous Network

Let us consider a typical scenario for the pre-5G Heterogeneous Network (HetNet), which is the physical foundation of redundant service placement at the edge. As demonstrated in Fig. 1, for a given region, the wireless infrastructure of the access network can be *simplified* into a Macro Base Station (MBS) and several Small-cell Base Stations (SBSs). The MBS is indispensable in any HetNet to provide ubiquitous coverage and support capacity, whose cell radius ranges from 8km to 30km. The SBSs, including femtocells, micro cells, and pico cells, are part of the network *densification* for densely populated urban areas.

Without loss of generality, WiFi access points, routers, and gateways are viewed as SBSs for simplification. Their cell radius ranges from 0.01km to 2km. The SBSs can be *logically* interconnected to transfer signaling, broadcast message, and select routes. It might be too luxurious if all SBSs are fully interconnected, and not necessarily achievable if they are set up by different Mobile Telecom Carriers (MTCs)<sup>2</sup>, but we reasonably assume that each SBS is mutually reachable to formulate an *undirected connected graph*. This can be seen in Fig. 1. Each SBS has a corresponding small-scale data-center attached for the deployment of microservices and the allocating of resources.

In this scenario, end users with their mobile devices can move arbitrarily within a certain range. For example, end users work within a building or rest at home. In this case, the connected SBS of each end user does not change.

## 2.2 Response Time of Microservices

A microservice-based application consists of multiple microservices. Each microservice can be executed by many available candidates. Take an arbitrary e-commerce application as an example. When we shop on a client browser, we firstly search the items we want, which can be realized by many site search APIs. Secondly, we add them to the cart and pay for them. The electronic payment can be accomplished by Alipay, WeChat Pay, or PayPal by invoking their APIs. After that, we can review and rate for those purchased items. In this example, each microservice is focused on single business capability. In addition, the considered application might have complex compositional structures and complex correlations between the fore-and-aft candidates because of *bundle sales*. For example, when we are shopping on Taobao<sup>3</sup>, only Alipay is supported for online payment. The application in the above example has a linear structure. As a beginning, this paper only cares about the *sequentially* composed application. In practice, a general directed acyclic graph (DAG) can be decomposed into several linear chains by applying Flow Decomposition Theorem (located in Chapter 03) [18]. We leave the extension to future work.

The pre-5G HetNet allows SBSs to share a mobile service provision platform, where user configurations and contextual information can be uniformly managed. As we have mentioned before, the unified platform can be implemented by Kubernetes. In our scenario, each mobile device sends its service request to the nearest SBS for the strongest signal of the established link. However, if there are no SBSs accessible, the request has to be responded by the MBS and processed by cloud data-centers. All the possibilities of the response status of the first microservice is discussed below.

- 1) The requested candidate is deployed at the chosen SBS. It will be processed by this SBS instantly.
- 2) The requested candidate is not deployed at the nearest SBS but accessible on other SBSs, which leads to multi-hop transfers between the SBSs until the request is responded by another SBS. That is,

2. Whether SBSs are logically connected comes down to their IP segments.

3. Taobao is the world's biggest e-commerce website, established in China.

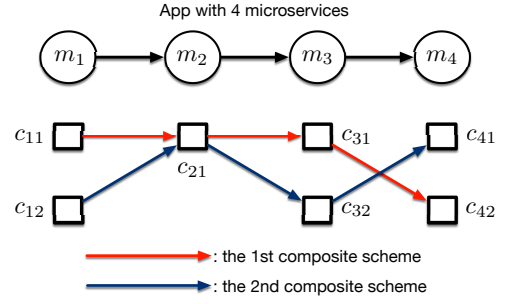


Fig. 2. Two service composition schemes for a 4-microservice app.

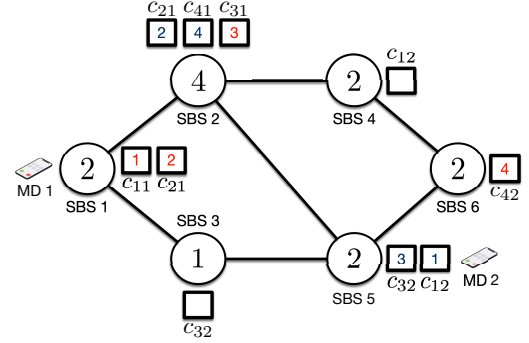


Fig. 3. The placement of each candidates on the HetNet.

the request will *route through* the HetNet until it is responded by an SBS who deploys the required candidate.

- 3) The requested candidate is not deployed on any SBSs in the HetNet. It can only be processed by cloud through backbone transmission.

For the subsequent microservices, the response status also faces many possibilities:

- 1) The previous candidate is processed by an SBS. Under this circumstance, for candidate of this microservice, if its instance can be found in the HetNet, multi-hop transfer is required. Otherwise, it has to be processed by cloud.
- 2) The previous candidate is processed by cloud. Under this circumstance, the candidates of subsequent microservices should always be responded by cloud without unnecessary backhaul.

Our job is to find an optimal redundant placement policy with the trade-offs between resource occupation and response time considered. We should know which candidates who might as well be redundant and where to deploy them.

## 2.3 A Working Example

This subsection describes a small-scale working example.

**Microservices and Candidates:** Fig. 2 demonstrates a chained application constitutive of four microservices. Each microservice has 2, 1, 2, and 2 candidates, respectively. The first service composition scheme is  $c_{11} \rightarrow c_{21} \rightarrow c_{31} \rightarrow c_{42}$ , and the second service composition scheme is  $c_{12} \rightarrow c_{21} \rightarrow c_{32} \rightarrow c_{41}$ . In practice, the composition scheme is decided by the daily usage habits of end users. It might be strongly

biased. Besides, because of bundle sales, part of the composition might be fixed. We will describe the composition in terms of a joint probability distribution in Subsection 3.1.

**Service Placement of Instances:** In Fig. 3, the undirected connected graph consists of 6 SBSs. The number tagged inside each SBS is its maximum number of placeable candidates. For example, SBS2 can be placed at most 4 candidates. This constraint exists because the edge sites have very limited computation and storage resources, compared with cloud data-centers. The squares beside each SBS are the deployed candidates. For example, SBS1 deploys two candidates,  $c_{11}$  and  $c_{21}$ . Notice that because of the redundancy mechanism, the same candidate can be deployed on multiple SBSs. For example,  $c_{21}$  is dispatched to both SBS1 and SBS2.

Fig. 3 also demonstrates two mobile devices, MD1 and MD2, which are located beside SBS1 and SBS5, respectively. It means that the SBSs closest to MD1 and MD2 are SBS1 and SBS5, respectively. As we have already mentioned in Subsection 2.2, the service request of each mobile device is responded by the nearest SBS. Thus, for MD1 and MD2, SBS1 and SBS5 are the corresponding SBS for responding, respectively.

**Response Time Calculation:** We assume that the service composition scheme of MD1 is the red one in Fig. 2, and MD2's is the blue one. The number tagged inside each candidate is the executing sequence. Let us take a closer look at MD1.

- 1)  $c_{11}$ : Because  $c_{11}$  is deployed on SBS1, the response time of  $c_{11}$  is equal to the sum of the expenditure of time on wireless access between MD1 and SBS1 and the processing time of  $c_{11}$  on SBS1.
- 2)  $c_{21}$ : Because  $c_{21}$  can also be found on SBS1, the expenditure time is zero. The response time of  $c_{21}$  consists of only the processing time of  $c_{21}$  on SBS1.
- 3)  $c_{31}$ :  $c_{31}$  can only be found on SBS2, thus the expenditure time of it is equal to the routing time from SBS1 to SBS2. In this paper, we assume that the routing between two nodes always selects the nearest path in the undirected graph. Thus, only one hop is required (SBS1  $\rightarrow$  SBS2). Thus, the response time of  $c_{31}$  consists of the routing time from SBS1 to SBS2 and the processing time of  $c_{31}$  on SBS2.
- 4)  $c_{42}$ :  $c_{42}$  can only be found on SBS6. Thus, the expenditure requires 2 hops (SBS2  $\rightarrow$  SBS4  $\rightarrow$  SBS6 or SBS2  $\rightarrow$  SBS5  $\rightarrow$  SBS6). Finally, the output of  $c_{42}$  need to be transferred back to MD1 via SBS1. The nearest path from SBS6 to SBS1 requires 3 hops. There are three alternatives: (i) SBS6  $\rightarrow$  SBS4  $\rightarrow$  SBS2  $\rightarrow$  SBS1 or (ii) SBS6  $\rightarrow$  SBS5  $\rightarrow$  SBS2  $\rightarrow$  SBS1 or (iii) SBS6  $\rightarrow$  SBS5  $\rightarrow$  SBS3  $\rightarrow$  SBS1. Thus, the response time of  $c_{42}$  consists of the routing time from SBS2 to SBS6, the processing time of  $c_{42}$  on SBS6, the routing time from SBS6 to SBS1, and the wireless transmission time from SBS1 to MD1.

The response time of the 1<sup>st</sup> composition scheme is the sum of the response time of  $c_{11}$ ,  $c_{21}$ ,  $c_{31}$ , and  $c_{42}$ . The same procedure applies to MD2. In addition, there are two unexpected cases need to be addressed. The first one is that if a mobile device is covered by no SBS, the response should

TABLE 1  
Summary of key notations.

Notation	Description
$M$	the number of SBSs, $M =  \mathcal{M} $
$N$	the number of mobile devices, $N =  \mathcal{N} $
$Q$	the number of microservices in the application, $Q =  \mathcal{Q} $
$t_q, q \in \mathcal{Q}$	the $q$ th microservice in the application
$\mathcal{M}_i$	the set of SBSs that can be connected by the $i$ th mobile device
$\mathcal{N}_j$	the set of mobile devices that can be connected by the $j$ th SBS
$C_q$	the number of candidates of the $q$ th microservice, $C_q =  \mathcal{C}_q , q \in \mathcal{Q}$
$s_q^c, c \in \mathcal{C}_q$	the $c$ th candidate of the $q$ th microservice
$\mathcal{D}(s_q^c)$	the set of SBSs on which $s_q^c$ is deployed
$E(s_q^c)$	the random event that for microservice $t_q$ , the $c$ th candidate is selected for execution
$j_i^*$	the nearest SBS to the $i$ th mobile device
$j_p(s_q^c)$	the SBS who actually processes the candidate $s_q^c$
$d(i, j)$	the Euclidean distance between the $i$ th mobile device and the $j$ th SBS
$\tau_{in}(s_q^c)$	the data uplink transmission time of the candidate $s_q^c$
$\tau_{exe}(j_p(s_q^c))$	the execution time of the candidate $s_q^c$ on the $j_p$ th SBS
$\zeta(j_1, j_2)$	the hop-count between the $j_1$ th SBS and the $j_2$ th SBS
$b_j$	the maximum number of microservice instances can be deployed on the $j$ th SBS

be made by the MBS and all the microservices are processed by cloud. The second one is that if a required candidate is not deployed on any SBS, then a communication link from the SBS processing the last candidate to the cloud should be established. This candidate and all the candidates of the rest microservices will be processed on cloud. The response time is calculated in a different way for these cases. Nevertheless, all the contingencies are taken into account in our system model, elaborated in Section 3.

Obviously, a better service placement policy can lead to less time spent. Our job is to find a service placement policy to minimize the response time of all mobile devices. What need to determine are not only how many instances required for each candidate, but also which edge sites to place them. The next section will demonstrate the rigorous formulation of system model.

### 3 SYSTEM MODEL

The HetNet consists of  $N$  mobile devices, indexed by  $\mathcal{N} \triangleq \{1, \dots, i, \dots, N\}$ ,  $M$  SBSs, indexed by  $\mathcal{M} \triangleq \{1, \dots, j, \dots, M\}$ , and one MBS, indexed by 0. Considering that each mobile device might be covered by many SBSs, let us use  $\mathcal{M}_i$  to denote the set of SBSs whose wireless signal covers the  $i$ th mobile devices. Correspondingly,  $\mathcal{N}_j$  denotes the set of mobile devices that are covered by the  $j$ th SBS. Notice that the service request from a mobile device is responded by its nearest available SBS, otherwise the MBS. The MBS here is to provide ubiquitous signal coverage and is always connectable to each mobile device. Both the SBSs and the MBS are connected to the backbone. Table 1 lists key notations in this paper.

### 3.1 Describing the Correlated Microservices

We consider an application with  $Q$  sequential composite microservices  $\langle t_1, \dots, t_Q \rangle$ , indexed by  $Q$ .  $\forall q \in Q$ , microservice  $t_q$  has  $C_q$  candidates  $\{s_q^1, \dots, s_q^c, \dots, s_q^{C_q}\}$ , indexed by  $C_q$ . Let us use  $\mathcal{D}(s_q^c) \subseteq \mathcal{M}$  to denote the set of SBSs on which  $s_q^c$  is deployed. Our redundancy mechanism allows that  $|\mathcal{D}(s_q^c)| > 1$ , which means one candidate instance could be dispatched to more than one SBS. Besides, let us use  $E(s_q^c), c \in C_q$  to represent that for microservice  $t_q$ , the  $c$ th candidate is selected for execution.  $E(s_q^c)$  can be viewed as a random event with an unknown distribution. Further,  $\mathbb{P}(E(s_q^c))$  denote the probability that  $E(s_q^c)$  happens. Thus, for each mobile device, its selected candidates can be described as a  $Q$ -tuple:

$$(\mathbf{s}) \triangleq \langle E(s_1^{c_1}), \dots, E(s_Q^{c_Q}) \rangle, \quad (1)$$

where  $q \in Q, c_q \in C_q$ .

The sequential composite application might have correlations between the fore-and-aft candidates. The definition below gives a rigorous mathematical description.

**Definition 3.1. Correlations of Composite Service**  $\forall q \in Q \setminus \{1\}$ ,  $c_1 \in C_{q-1}$ , and  $c_2 \in C_q$ , candidate  $s_q^{c_2}$  and  $s_{q-1}^{c_1}$  are correlated iff  $\mathbb{P}(E(s_q^{c_2})|E(s_{q-1}^{c_1})) \equiv 1$ , and  $\forall c_2 \in C_q \setminus \{c_2\}, \mathbb{P}(E(s_q^{c_2})|E(s_{q-1}^{c_1})) \equiv 0$ .

With the above definition, the probability  $\mathbb{P}(E(\mathbf{s}))$  can be calculated by

$$\mathbb{P}(E(\mathbf{s})) = \prod_{q=2}^Q \mathbb{P}(E(s_q^{c_q})|E(s_{q-1}^{c_{q-1}})). \quad (2)$$

### 3.2 Calculating the Response Time

The response time of one candidate consists of data uplink transmission time, service execution time, and data downlink transmission time. The data uploaded is mainly encoded service requests and configurations while the output is mainly the feedback on successful service execution or a request to invoke the next candidate. If all requests are responded within the access network, most of the time is spent on routing with multi-hops between SBSs. Notice that except the last one, each candidate's data uplink transmission time is equal to the data downlink transmission time of the candidate of its previous microservice.

#### 3.2.1 For the Initial Candidate

For the  $i$ th mobile device and its selected candidate  $s_1^{c_1}(i)$  of the initial microservice  $t_1$ , where  $c_1 \in C_1$ , (I) if the  $i$ th mobile device is not covered by any SBSs around, i.e.,  $\mathcal{M}_i = \emptyset$ , the request has to be responded by the MBS and processed by cloud data-center. (II) If  $\mathcal{M}_i \neq \emptyset$ , as we have mentioned before, the nearest SBS  $j_i^* \in \mathcal{M}_i$  is chosen and connected. Under this circumstance, a classified discussion is required.

- 1) If the candidate  $s_1^{c_1}(i)$  is not deployed on any SBSs from  $\mathcal{M}$ , i.e.  $\mathcal{D}(s_1^{c_1}(i)) = \emptyset$ , the request still has to be forwarded to cloud data-center through backbone transmission.
- 2) If  $\mathcal{D}(s_1^{c_1}(i)) \neq \emptyset$  and  $j_i^* \in \mathcal{D}(s_1^{c_1}(i))$ , the request can be directly processed by SBS  $j_i^*$  without any hops.

- 3) If  $\mathcal{D}(s_1^{c_1}(i)) \neq \emptyset$  and  $j_i^* \notin \mathcal{D}(s_1^{c_1}(i))$ , the request has to be responded by  $j_i^*$  and processed by another SBS from  $\mathcal{D}(s_1^{c_1}(i))$ .

We use  $j_p(s)$  to denote the SBS who actually processes  $s$ . Remember that the routing between two nodes always selects the nearest path. Thus, for  $q = 1$ ,  $j_p(s_1^{c_1}(i))$  can be obtained by (3), where  $\zeta(j_1, j_2)$  is the shortest number of hops from node  $j_1$  to node  $j_2$ .

In this paragraph, we calculate the response time of  $s_1^{c_1}(i)$ . We use  $d(i, j)$  to denote the reciprocal of the bandwidth between  $i$  and  $j$ . Obviously, the expenditure of time on wireless access is inversely proportional to the bandwidth of the link. Besides, the expenditure of time on routing is directly proportional to the number of hops between the source node and destination node. As a result, the data uplink transmission time  $\tau_{in}(s_1^{c_1}(i))$  is summarized in (4), where  $\tau_b$  is the time on backbone transmission,  $\alpha$  is size of input data stream from each mobile device to the initial candidate (measured in bits), and  $\beta$  is a positive constant representing the rate of wired link between SBSs. We use  $\tau_{exe}(j_p(s_1^{c_1}(i)))$  to denote the microservice execution time on the SBS  $j_p$  for the candidate  $s_1^{c_1}(i)$ . The data downlink transmission time is the same as the uplink time of the next microservice, which is discussed hereinafter.

#### 3.2.2 For the Intermediate Candidates

For the  $i$ th mobile device and its selected candidate  $s_q^{c_q}(i)$  of microservice  $t_q$ , where  $q \in Q \setminus \{1, Q\}$ ,  $c_q \in C_q$ , the analysis of its data uplink transmission time is correlated with  $j_p(s_{q-1}^{c_{q-1}}(i))$ , i.e. the SBS who processes  $s_{q-1}^{c_{q-1}}(i)$ .  $\forall q \in Q \setminus \{1\}$ , the calculation of  $j_p(s_q^{c_q}(i))$  is summarized in (5). This formula is closely related to (3).

In this paragraph, we calculate the response time of  $s_q^{c_q}(i)$ . (I) If  $j_p(s_{q-1}^{c_{q-1}}(i)) \in \mathcal{D}(s_q^{c_q}(i))$ , then the data downlink transmission time of previous candidate  $s_{q-1}^{c_{q-1}}(i)$ , which is also the data uplink transmission time of current candidate  $s_q^{c_q}(i)$ , is zero. That is,  $\tau_{out}(s_{q-1}^{c_{q-1}}(i)) = \tau_{in}(s_q^{c_q}(i)) = 0$ . It is because both  $s_{q-1}^{c_{q-1}}(i)$  and  $s_q^{c_q}(i)$  are deployed on the SBS  $j_p(s_{q-1}^{c_{q-1}}(i))$ , which leads to the number of hops being zero. (II) If  $j_p(s_{q-1}^{c_{q-1}}(i)) \notin \mathcal{D}(s_q^{c_q}(i))$ , a classified discussion has to be launched.

- 1) If  $j_p(s_{q-1}^{c_{q-1}}(i)) = \text{cloud}$ , which means the request of  $t_{q-1}$  from the  $i$ th mobile device is responded by cloud data-center. In this case, the invocation for  $t_q$  can be directly processed by cloud without backhaul. Thus, the data uplink transmission time of  $t_q$  is zero, too.
- 2) If  $j_p(s_{q-1}^{c_{q-1}}(i)) \neq 0$  and  $\mathcal{D}(s_q^{c_q}(i)) = \emptyset$ , which means  $s_q^{c_q}(i)$  is not deployed on any SBSs in the HetNet. As a result, the invocation for  $t_q$  has to be responded by cloud data-center through backbone transmission.
- 3) If  $j_p(s_{q-1}^{c_{q-1}}(i)) \neq 0$ ,  $\mathcal{D}(s_q^{c_q}(i)) \neq \emptyset$  but  $j_p(s_{q-1}^{c_{q-1}}(i)) \notin \mathcal{D}(s_q^{c_q}(i))$ , which means both  $s_{q-1}^{c_{q-1}}(i)$  and  $s_q^{c_q}(i)$  are processed by the SBSs in the HetNet but not the same one. In this case, we can calculate the data uplink transmission time by finding the shortest path from  $j_p(s_{q-1}^{c_{q-1}}(i))$  to a SBS in  $\mathcal{D}(s_q^{c_q}(i))$ .

The above analysis is summarized in (6).

$$j_p(s_1^{c_1}(i)) = \begin{cases} \text{cloud}, & \mathcal{M}_i = \emptyset \text{ or } \mathcal{D}(s_1^{c_1}(i)) = \emptyset; \\ j_i^*, & \mathcal{D}(s_1^{c_1}(i)) \neq \emptyset, j_i^* \in \mathcal{D}(s_1^{c_1}(i)); \\ \operatorname{argmin}_{j^* \in \mathcal{D}(s_1^{c_1}(i))} \zeta(j_i^*, j^*), & \text{otherwise} \end{cases} \quad (3)$$

$$\tau_{in}(s_1^{c_1}(i)) = \begin{cases} \alpha \cdot d(i, 0) + \tau_b, & \mathcal{M}_i = \emptyset; \\ \alpha \cdot d(i, j_i^*) + \tau_b, & \mathcal{M}_i \neq \emptyset, \mathcal{D}(s_1^{c_1}(i)) = \emptyset; \\ \alpha \cdot d(i, j_i^*), & \mathcal{M}_i \neq \emptyset, j_i^* \in \mathcal{D}(s_1^{c_1}(i)); \\ \alpha \cdot d(i, j_i^*) + \beta \cdot \min_{j^* \in \mathcal{D}(s_1^{c_1}(i))} \zeta(j_i^*, j^*), & \text{otherwise} \end{cases} \quad (4)$$

$$j_p(s_q^{c_q}(i)) = \begin{cases} \text{cloud}, & \mathcal{M}_i = \emptyset \text{ or } \mathcal{D}(s_q^{c_q}(i)) = \emptyset; \\ j_p(s_{q-1}^{c_{q-1}}(i)), & \mathcal{D}(s_q^{c_q}(i)) \neq \emptyset, j_p(s_{q-1}^{c_{q-1}}(i)) \in \mathcal{D}(s_q^{c_q}(i)); \\ \operatorname{argmin}_{j^* \in \mathcal{D}(s_q^{c_q}(i))} \zeta(j_p(s_{q-1}^{c_{q-1}}(i)), j^*), & \text{otherwise} \end{cases} \quad (5)$$

$$\tau_{in}(s_q^{c_q}(i)) = \begin{cases} 0, & j_p(s_{q-1}^{c_{q-1}}(i)) \in \mathcal{D}(s_q^{c_q}(i)) \text{ or } j_p(s_{q-1}^{c_{q-1}}(i)) = \text{cloud}; \\ \tau_b, & j_p(s_{q-1}^{c_{q-1}}(i)) \neq 0, \mathcal{D}(s_q^{c_q}(i)) = \emptyset; \\ \beta \cdot \min_{j^* \in \mathcal{D}(s_q^{c_q}(i))} \zeta(j_p(s_{q-1}^{c_{q-1}}(i)), j^*), & \text{otherwise} \end{cases} \quad (6)$$

$$\tau_{out}(s_Q^{c_Q}(i)) = \begin{cases} \tau_b + \alpha \cdot d(i, 0), & j_p(s_Q^{c_Q}(i)) = \text{cloud}; \\ \beta \cdot \zeta(j_p(s_Q^{c_Q}(i)), j_i^*) + \alpha \cdot d(i, j_i^*), & \text{otherwise} \end{cases} \quad (7)$$

### 3.2.3 For the Last Candidate

For the  $i$ th mobile device and its selected candidate  $s_Q^{c_Q}(i)$  of the last microservice  $t_Q$ , where  $c_Q \in \mathcal{C}_Q$ , the data uplink transmission time  $\tau_{in}(s_Q^{c_Q}(i))$  is also calculated by (6), with every  $q$  replaced by  $Q$ . However, for the data downlink transmission time  $\tau_{out}(s_Q^{c_Q}(i))$ , a classified discussion is required: (I) If  $j_p(s_Q^{c_Q}(i)) = 0$ , which means the chosen candidate of the last microservice is processed by cloud data-center, the processed result need to be returned from cloud to the  $i$ th mobile device through backhaul transmission<sup>4</sup>. (II) If  $j_p(s_Q^{c_Q}(i)) \neq 0$ , which means the chosen candidate of the last microservice is processed by a SBS in the HetNet. In this case, the result should be delivered to the  $i$ th mobile device via  $j_p(s_Q^{c_Q}(i))$  and  $j_i^*$ <sup>5</sup>. (7) summarizes the calculation of  $\tau_{out}(s_Q^{c_Q}(i))$ .

Based on the above analysis, the response time of the  $i$ th mobile device is

$$\tau(E(s(i))) = \sum_{q=1}^Q \left( \tau_{in}(s_q^{c_q}(i)) + \tau_{exe}(j_p(s_q^{c_q}(i))) \right) + \tau_{out}(s_Q^{c_Q}(i)). \quad (8)$$

So far, the system model has been elaborated. The assumptions in this paper are summarized as follows.

1) We assume that the edge sites can form an undirected connected graph. In MEC, this assumption is rational and frequently-used, especially for the research on Network Slicing [19].

2) We assume that it is the nearest SBS that responses to the initial microservice of a mobile device. This assumption is naive and widely-used because it leads to the minimal first-step communication cost.

4. We assume that the backhaul can only be transferred through the MBS.

5.  $j_p(s_Q^{c_Q}(i))$  and  $j_i^*$  could be the same one. In this case, the number of hops between them is zero.

3) We only consider the composite application with linear structure. As we have mentioned, a general DAG can be decomposed into several linear chains, thus we leave the extension to future work.

4) We assume that the expenditure of time on routing is directly proportional to the number of hops. In MEC, the HetNet is a given region whose range is within tens of kilometers. The transmission time is mainly spent on routing and transit. Thus, this assumption is rational.

5) We assume that the backhaul can only be transferred through the MBS. There are multiple alternatives for backhaul in 5G communications. We make the assumption to simplify the problem formulation for the chosen candidate of the last microservice.

### 3.3 Problem Formulation

Our job is to find an optimal redundant placement policy to minimize the overall latency under the limited capability of SBSs. The heterogeneity of edge sites is directly embodied in the number of *can-be-deployed* candidates. Let us use  $b_j$  to denote this number for the  $j$ th SBS. In the heterogenous edge,  $b_j$  could vary considerably. The following constraint should be satisfied:

$$\sum_{q \in \mathcal{Q}} \sum_{c \in \mathcal{C}_q} \mathbb{1}\{j \in \mathcal{D}(s_q^{c_q})\} \leq b_j, \forall j \in \mathcal{M}, \quad (9)$$

where  $\mathbb{1}\{\cdot\}$  is the indicator function. Finally, the optimal placement problem can be formulated as:

$$\begin{aligned} \mathcal{P}_1 : \min_{\mathcal{D}(s_q^{c_q})} & \sum_{i=1}^N \tau(E(s(i))) \\ \text{s.t.} & \quad (9), \end{aligned}$$

where the decision variables are  $\mathcal{D}(s_q^{c_q}(i)), \forall q \in \mathcal{Q}, c \in \mathcal{C}_q$ , and the optimization goal is the sum of response time of all mobile devices.



## 4 ALGORITHM DESIGN

In this section, we elaborate our algorithm for  $\mathcal{P}_1$ . Firstly, we recode the decision variables as  $\mathbf{x}$  to *shrink* the size of feasible region. Based on that, we propose the SAA-RP framework. It includes a subroutine, named Genetic Algorithm-based Server Selection (GASS) algorithm. The details are presented as follows.

### 4.1 Variable Recoding

Let us use  $\mathbf{W}(i) \triangleq (\text{canIdx}(t_1), \dots, \text{canIdx}(t_Q))$  to denote the random vector on the chosen service composition scheme of the  $i$ th mobile device, where  $\text{canIdx}(t_q)$  returns the index of the chosen candidate of the  $q$ th microservice.  $\mathbf{W}(i)$  and  $E(s(i))$  describe the same thing from different perspectives. However, the former is more concise. Then, we use  $\mathbf{W} \triangleq (\mathbf{W}(1), \dots, \mathbf{W}(N))$  to denote the global random vector by taking all mobile devices into account. Let us use  $\mathbf{x} \triangleq [\mathbf{x}(b_1), \dots, \mathbf{x}(b_M)]^\top$  to recode the global *deploy-or-not* vector.  $\forall j \in \mathcal{M}, \mathbf{x}(b_j)$  is a deployment vector for SBS  $j$ , whose length is  $b_j$ . By doing this, the constraint (9) can be removed because it is reflected in how  $\mathbf{x}$  encodes.  $\forall j \in \mathcal{M}$ , each element of  $\mathbf{x}(b_j)$  is chosen from  $\{0, 1, \dots, \sum_{q=1}^Q C_q\}$ , i.e. the global index of each candidate. That is, any candidate can appear in any number of SBSs in the HetNet. Thus, the redundancy mechanism is also reflected in how  $\mathbf{x}$  encodes.  $\forall j \in \mathcal{M}, \mathbf{x}(b_j) = \mathbf{0}$  means that the  $j$ th SBS does not deploy any candidate.

$\mathbf{x}$  is a new encoding of  $\mathcal{D}(s_q^c(i))$ . As a result, we can reconstitute  $\tau(E(s(i)))$  as  $\tau(\mathbf{x}, \mathbf{W}(i))$ . As such, the optimization goal can be written as

$$g(\mathbf{x}) \triangleq \mathbb{E}[G(\mathbf{x}, \mathbf{W})] = \mathbb{E}\left[\sum_{i=1}^N \tau(\mathbf{x}, \mathbf{W}(i))\right], \quad (10)$$

and the optimal placement problem is

$$\mathcal{P}_2 : \min_{\mathbf{x} \in \mathcal{X}} g(\mathbf{x}).$$

$\mathcal{P}_2$  is a stochastic discrete optimization problem with independent variable  $\mathbf{x}$ , where  $\mathcal{X}$  is the feasible region.  $\mathcal{X}$ , although finite, is very large. Therefore, enumeration approach is inadvisable. Besides, the problem has an uncertain random vector  $\mathbf{W}$  with probability distribution  $\mathbb{P}(E(s(i)))$ .

### 4.2 The SAA-RP Framework

Let us take a closer look at  $\mathcal{P}_2$ . Firstly, the random vector  $\mathbf{W}$  is *exogenous* because the decision on  $\mathbf{x}$  does not affect the distribution of  $\mathbf{W}$ . Secondly, for a given  $\mathbf{W}$ ,  $G(\mathbf{x}, \mathbf{W})$  can be easily evaluated for any  $\mathbf{x}$ . Thus, the observation of  $G(\mathbf{x}, \mathbf{W})$  is *constructive*. As a result, we can apply the Sample Average Approximation (SAA) approach to  $\mathcal{P}_1$  [20] to handle with the uncertainty.

SAA is a classic Monte Carlo simulation-based method. In the following section, we elaborate how we apply the SAA method to  $\mathcal{P}_2$ . Formally, we define the SAA problem  $\mathcal{P}_3$ . Let  $\mathbf{W}^1, \mathbf{W}^2, \dots, \mathbf{W}^R$  be an independently and identically distributed (i.i.d.) random sample of  $R$  realizations of the random vector  $\mathbf{W}$ . The SAA function is defined as

$$\hat{g}_R(\mathbf{x}) \triangleq \frac{1}{R} \sum_{r=1}^R G(\mathbf{x}, \mathbf{W}^r), \quad (11)$$

and the SAA problem  $\mathcal{P}_3$  is defined as

$$\mathcal{P}_3 : \min_{\mathbf{x} \in \mathcal{X}} \hat{g}_R(\mathbf{x}).$$

By Monte Carlo Sampling, with support from the *Law of Large Numbers* [21], when  $R$  is large enough, the optimal value of  $\hat{g}_R(\mathbf{x})$  can converge to the optimal value of  $g(\mathbf{x})$  with probability one (w.p.1). As a result, we only need to care about how to solve  $\mathcal{P}_3$  as optimal as possible.

---

#### Algorithm 1 SAA-based Redundant Placement (SAA-RP)

---

- 1: Choose initial sample size  $R$  and  $R'$  ( $R' \gg R$ )
  - 2: Choose the number of replications  $L$  (indexed by  $\mathcal{L}$ )
  - 3: Set up a gap tolerance  $\epsilon$
  - 4: **for**  $l = 1$  to  $L$  **in parallel do**
  - 5:   Generate  $R$  independent samples  $\mathbf{W}_l^1, \dots, \mathbf{W}_l^R$
  - 6:   Call GASS to obtain the minimum value of  $\hat{g}_R(\mathbf{x}_l)$  with the form of  $\frac{1}{R} \sum_{r=1}^R G(\mathbf{x}_l, \mathbf{W}_l^r)$
  - 7:   Record the optimal goal  $\hat{g}_R(\hat{\mathbf{x}}_l^*)$  and the corresponding variable  $\hat{\mathbf{x}}_l^*$  returned from GASS
  - 8: **end for**
  - 9:  $\bar{v}_R^* \leftarrow \frac{1}{L} \sum_{l=1}^L \hat{g}_R(\hat{\mathbf{x}}_l^*)$
  - 10: **for**  $l = 1$  to  $L$  **in parallel do**
  - 11:   Generate  $R'$  independent samples  $\mathbf{W}_l^1, \dots, \mathbf{W}_l^{R'}$
  - 12:    $v_{R'}^l \leftarrow \frac{1}{R'} \sum_{r'=1}^{R'} G(\hat{\mathbf{x}}_l^*, \mathbf{W}_l^{r'})$
  - 13: **end for**
  - 14: Get the worst replication  $v_{R'}^* \leftarrow \max_{l \in \mathcal{L}} v_{R'}^l$
  - 15: **if** the gap  $v_{R'}^* - \bar{v}_R^* < \epsilon$  **then**
  - 16:   Choose the best solution  $\hat{\mathbf{x}}_l^*$  among all  $L$  replications
  - 17: **else**
  - 18:   Increase  $R$  (for drill) and  $R'$  (for evaluation)
  - 19:   **goto** Step. 4
  - 20: **end if**
  - 21: **return** the best solution  $\hat{\mathbf{x}}_l^*$
- 

The SAA-RP framework is presented in **Algorithm 1**. Firstly, we need to select the sample size  $R$  and  $R'$  appropriately. As the sample size  $R$  increases, the optimal solution of the SAA problem  $\mathcal{P}_2$  converges to its *true problem*  $\mathcal{P}_1$ . However, the computational complexity for solving  $\mathcal{P}_2$  increases at least linearly, even exponentially, in the sample size  $R$  [20]. Therefore, when we choose  $R$ , the trade-off between the quality of the optimality and the computation effort should be taken into account. Besides,  $R'$  here is used to obtain an estimate of  $\mathcal{P}_1$  with the obtained solution of  $\mathcal{P}_2$ . In order to obtain an accurate estimate, we have every reason to choose a relatively large sample size  $R'$  ( $R' \gg R$ ). Secondly, inspired by [20], we replicate generating and solving  $\mathcal{P}_2$  with  $L$  i.i.d. replications. From Step. 4 to Step. 8, we call the algorithm GASS to obtain the asymptotically optimal solution of  $\mathcal{P}_2$  and record the best-so-far results. From Step. 10 to Step. 13, we estimate the true value of  $\mathcal{P}_1$  for each replication. After that, those estimates are compared with the average of those optimal solutions of  $\mathcal{P}_2$ . If the maximum gap is smaller than the tolerance, SAA-RP returns the best solution among  $L$  replications and the algorithm terminates, otherwise we increase  $R$  and  $R'$  and drill again.

### 4.3 The GASS Algorithm

GASS is implemented based on the well-known Genetic Algorithm (GA). The detailed procedure is demonstrated in

**Algorithm 2.** Firstly, we initialize the necessary parameters, including the population size  $P$ , the number of iterations  $it$ , and the probability of crossover  $\mathbb{P}_c$  and mutation  $\mathbb{P}_m$ , respectively. After that, we randomly generate the initial population from the domain  $\mathcal{X}$ . From Step. 6 to Step. 10, GASS executes the crossover operation. At the beginning of this operation, GASS checks whether crossover need to be executed. If yes, GASS choose the *best* two chromosomes according to their fitness values. With that, the latter part of  $\mathbf{x}_{p_1}$  and  $\mathbf{x}_{p_2}$  are exchanged since the position  $\mathbf{x}(b_j)$ . From Step. 11 to Step. 13, GASS executes the mutation operation. At the beginning of this operation, it checks whether each chromosome can mutate according to the mutation probability  $\mathbb{P}_m$ . At the end, only the chromosome with the best fitness value can be returned.

---

**Algorithm 2** GA-based Server Selection (GASS)

---

- 1: Initialize the population size  $P$ , number of iterations  $it$ , the probability of crossover  $\mathbb{P}_c$  and mutation  $\mathbb{P}_m$
- 2: Randomly generate  $P$  chromosomes  $\mathbf{x}_1, \dots, \mathbf{x}_P \in \mathcal{X}$
- 3: **for**  $t = 1$  to  $it$  **do**
- 4:    $\forall p \in \{1, \dots, P\}$ , renew the optimization goal of  $\mathcal{P}_2$ , i.e.  $\hat{g}_R(\mathbf{x}_p)$ , according to (11)
- 5:   **for**  $p = 1$  to  $P$  **do**
- 6:     **if**  $\text{rand}() < \mathbb{P}_c$  **then**
- 7:       Choose two chromosomes  $p_1$  and  $p_2$  according to the probability distribution:

$$\mathbb{P}(p \text{ is chosen}) = \frac{1/\hat{g}_R(\mathbf{x}_p)}{\sum_{p'=1}^P 1/\hat{g}_R(\mathbf{x}_{p'})}$$

- 8:       Randomly choose SBS  $j \in \mathcal{M}$
  - 9:       Crossover the segment of  $\mathbf{x}_{p_1}$  and  $\mathbf{x}_{p_2}$  after the partitioning point  $\mathbf{x}(b_{j-1})$ :
 
$$[\mathbf{x}_{p_1}(b_j), \dots, \mathbf{x}_{p_1}(b_M)] \leftrightarrow [\mathbf{x}_{p_2}(b_j), \dots, \mathbf{x}_{p_2}(b_M)]$$
  - 10:      **end if**
  - 11:     **if**  $\text{rand}() < \mathbb{P}_m$  **then**
  - 12:       Randomly choose SBS  $j \in \mathcal{M}$  and re-generate the segment  $\mathbf{x}_p(b_j)$
  - 13:     **end if**
  - 14:   **end for**
  - 15: **end for**
  - 16: **return**  $\text{argmin}_p \hat{g}(\mathbf{x}_p)$  from  $P$  chromosomes
- 

#### 4.4 Strength and Advantages

This subsection summarizes the strength and advantages of SAA-RP and GASS.

1) Notice that SAA-RP is not deployed online. It can be periodically re-run to follow up end users' service demand pattern. During each period, for example, a month or a quarter, end users' microservice composition preferences can be collected (under authorization of privacy). Pods can be reconstructed based on the result from SAA-RP. This process can be carried out through rolling upgrade with Kubernetes.

2) With the recoded decision variable  $\mathbf{x}$ , GASS is simple to operate and it enjoys a fast convergence rate. In the domain of  $\mathcal{P}_1$ , the number of elements is  $\exp\{\sum_{q \in \mathcal{Q}} C_q \cdot \ln M\}$ ,

which *exponentially* increases with the scale of microservices. However, after re-encoding, the number of elements in domain  $\mathcal{X}$  is  $\prod_{j \in \mathcal{M}} (b_j(\sum_{q \in \mathcal{Q}} C_q + 1) - \frac{b_j}{2}(b_j - 1))$ , which increases *polynomially* with the scale of microservices. The conclusion will be proved in Subsection 5.2 and verified in Subsection 6.3.2.

3) SAA-RP makes up with the shortcoming of the default scheduler of Kubernetes when encountering the MEC.. *Kube-scheduler* is a component responsible for the deployment of configured pods and microservices, which selects the node for a microservice instance in a two-step operation: Filtering and Scoring [16]. The filtering step finds the set of nodes who are feasible to schedule the microservice instance based on their available resources. In the scoring step, the *kube-scheduler* ranks the schedulable nodes to choose the most suitable one for the placement of the microservice instance. It places microservices only based on resources occupancy of nodes. By contrast, SAA-RP takes both the service request pattern of end users and the heterogeneity of the distributed nodes into consideration. SAA-RP makes a progress towards the resource orchestration on the heterogeneous edge.

## 5 THEORETICAL ANALYSIS

In this section, we analyze the optimality of SAA-RP and the complexity of GASS.

### 5.1 Solution Optimality

Recall that the domain  $\mathcal{X}$  of problem  $\mathcal{P}_2$  and  $\mathcal{P}_3$  is finite, whose size is  $\prod_{j \in \mathcal{M}} b_j(\sum_{q \in \mathcal{Q}} C_q + 1 - \frac{1}{2}(b_j - 1))$ . Thus,  $\mathcal{P}_2$  and  $\mathcal{P}_3$  have nonempty set of optimal solutions, denoted as  $\mathcal{X}^*$  and  $\hat{\mathcal{X}}_R$ , respectively. We let  $v^*$  and  $\hat{v}_R$  denote the optimal values of  $\mathcal{P}_2$  and  $\mathcal{P}_3$ , respectively. To analysis the optimality, we also define the set of  $\epsilon$ -optimal solutions.

**Definition 5.1.  $\epsilon$ -optimal Solutions** For  $\epsilon \geq 0$ , if  $\mathbf{x} \in \mathcal{X}$  and  $g(\mathbf{x}) \leq v^* + \epsilon$ , then we say that  $\mathbf{x}$  is an  $\epsilon$ -optimal solution of  $\mathcal{P}_1$ . Similarly, if  $\mathbf{x} \in \mathcal{X}$  and  $g(\mathbf{x}) \leq \hat{v}_R + \epsilon$ ,  $\mathbf{x}$  is an  $\epsilon$ -optimal solution of  $\mathcal{P}_3$ .

We use  $\mathcal{X}^\epsilon$  and  $\hat{\mathcal{X}}_R^\epsilon$  to denote the set of  $\epsilon$ -optimal solutions of  $\mathcal{P}_2$  and  $\mathcal{P}_3$ , respectively. Then we have the following proposition.

**Proposition 5.1.**  $\hat{v}_R \rightarrow v^*$  w.p.1 as  $R \rightarrow \infty$ ;  $\forall \epsilon \geq 0$ , the event  $\{\hat{\mathcal{X}}_R^\epsilon \rightarrow \mathcal{X}^\epsilon\}$  happens w.p.1 for  $R$  large enough.

**Proof 5.1.** It can be directly obtained from *Proposition 2.1* of [20], not tired in words here.

Proposition 5.1 implies that for almost every realization  $\omega = \{\mathbf{W}^1, \mathbf{W}^2, \dots\}$  of the random vector, there exists an integer  $R(\omega)$  such that  $\hat{v}_R \rightarrow v^*$  and  $\{\hat{\mathcal{X}}_R^\epsilon \rightarrow \mathcal{X}^\epsilon\}$  happen for all samples  $\{\mathbf{W}^1, \dots, \mathbf{W}^r\}$  from  $\omega$  with  $r \geq R(\omega)$ .

The following proposition demonstrates the convergence rate of SAA method.

**Proposition 5.2.**  $\forall \epsilon > 0$  small enough and  $\varpi \in [0, \epsilon)$ , for the probability  $\mathbb{P}(\{\hat{\mathcal{X}}_R^\varpi \subset \mathcal{X}^\epsilon\})$  to be at least  $1 - \alpha$ , the number of sample size  $R$  should satisfy

$$R \geq \frac{3\sigma_{max}^2}{(\epsilon - \varpi)^2} \cdot \sum_{j \in \mathcal{M}} \log \left( \frac{b_j}{\alpha} \cdot \left( \sum_{q \in \mathcal{Q}} C_q + \frac{3 - b_j}{2} \right) \right),$$



where

$$\sigma_{max}^2 \triangleq \max_{\mathbf{x} \in \mathcal{X} \setminus \mathcal{X}^\epsilon} \text{Var}[G(u(\mathbf{x}), \mathbf{W}) - G(\mathbf{x}, \mathbf{W})],$$

and  $u(\cdot)$  is a mapping from  $\mathcal{X} \setminus \mathcal{X}^\epsilon$  into  $\mathcal{X}^*$ , which satisfies that  $\forall \mathbf{x} \in \mathcal{X} \setminus \mathcal{X}^\epsilon, g(u(\mathbf{x})) \leq g(\mathbf{x}) - v^*$ .

**Proof 5.2.** It can be directly obtained by combining Proposition 2.2 of [20] with the size of  $\mathcal{X}$ . For more details, please consult [20] directly.

Proposition 5.2 implies that the number of samples  $R$  depends *logarithmically* on the feature of domain  $\mathcal{X}$  and the tolerance probability  $\alpha$ .

## 5.2 Algorithm Complexity

Obviously, GASS works through a mechanism of decomposition and reassembly. The following proposition holds.

**Proposition 5.3.** For problem  $\mathcal{P}_2$  and  $\mathcal{P}_3$ , when the number of SBSs is fixed as  $M$ , no matter how many mobile devices there are, the convergence time of GASS is  $O\left(\sqrt{R \cdot \prod_{j \in \mathcal{M}} (b_j(\sum_{q \in \mathcal{Q}} C_q + 1) - \frac{b_j}{2}(b_j - 1))}\right)$ .

**Proof 5.3.** Notice that each candidate only has to be deployed once on a single SBS. Thus, the number of possible values of  $\mathbf{x}(b_j)$  is  $b_j(\sum_{q \in \mathcal{Q}} C_q + 1) - \frac{b_j}{2}(b_j - 1)$ , and the number of elements in domain  $\mathcal{X}$ , i.e. the problem size, is  $R \cdot \prod_{j \in \mathcal{M}} (b_j(\sum_{q \in \mathcal{Q}} C_q + 1) - \frac{b_j}{2}(b_j - 1))$ . It follows from the fact that the  $k$ th element of  $\mathbf{x}(b_j)$  has  $\sum_{q \in \mathcal{Q}} C_q + 2 - k$  choices. Considering the processing building blocks of  $\mathcal{P}_3$  is of equal salience, the result can be obtained directly [22] [23].

Proposition 5.3 indicates that the complexity of GASS increases polynomially with the scale of the application, i.e.,  $\sum_{q \in \mathcal{Q}} C_q$ .

## 6 EXPERIMENTAL EVALUATION

In this section, we verify the superiority of the proposed algorithms through simulations.

### 6.1 Benchmark Policies

Our method is compared with several representative baselines and a state-of-the-art algorithm, GenDoc [12]. The baselines are performed in two scenarios while GenDoc is performed as it was defined in [12]. In the first scenario, redundancy is not allowed. Each candidate can only be dispatched to only one SBS. It is used to evaluate the superiority of redundant placement. In the second scenario, redundancy is allowed. It is used to evaluate the optimality of GASS. Those benchmark policies, including GenDoc, are used to replace GASS to generate the best-so-far solution of each sampling. In both scenarios, those benchmark policies will be run  $R$  times and the average value is returned. Details are summarized as follows.

(1) **Random Placement in Scenario #1 (RP1):**  $\forall q \in \mathcal{Q}, \forall c_q \in \mathcal{C}_q$ , dispatch  $c_q$  to a randomly chosen SBS  $j$  and decrement  $b_j$ . The procedure terminates if no available SBS.

(2) **Random Placement in Scenario #2 (RP2):**  $\forall q \in \mathcal{Q}, \forall c_q \in \mathcal{C}_q$ , randomly dispatch  $c_q$  to  $m$  SBSs. The number  $m \in \{m' \in \mathbb{N} | 0 \leq m' \leq M\}$  is generated randomly.

After that, for those SBSs, decrement their  $b_j$ . The procedure terminates if no available SBS.

(3) **Genetic Algorithm in Scenario #1 (GA1):** The chromosome is encoded as  $[p(s_1^1), \dots, p(s_Q^{C_Q})]^\top$ , where  $p(s)$  is the chosen SBS for the placement of the candidate  $s$ . This encoding ensures that each candidate can only be dispatched to one SBS. Based on that, each generation of chromosomes are created by selection, recombination, and mutation.

(4) **Greedy Placement in Scenario #2 (GP2):** For each SBS  $j \in \mathcal{M}$ ,  $b_j$  candidates will be deployed on it. It means that the feasible region lies in the boundary of the constraint (9). In each iteration, each end user always chooses the nearest available SBS to execute its selected candidates.

(5) **GenDoc:** GenDoc is a configuration-aware placement and scheduling algorithm, proposed in [12]. To apply GenDoc to our system model,  $\forall j \in \mathcal{M}$ , we set  $C_j^{vir} = b_j$ , where  $C_j^{vir}$  is the maximum virtual capacity of the  $j$ th SBS. More details can be found in Section 4.2 of [12].

### 6.2 Experimental Settings

All the experiments are implemented in MATLAB R2019b on macOS Catalina equipped with 3.1 GHz Quad-Core Intel Core i7 and 16 GB RAM. The parameter settings are discussed as follows.

TABLE 2  
Parameter settings.

Parameter	Value	Parameter	Value
$Q$	10	$\forall q, C_q$	[2, 5]
$N$	500	$M$	40
$b_l$	3	$b_u$	5
$\alpha$	[1, 8] kbits	$\beta$	5 ms
$\tau_b$	0.1 s	$L$	5
$\forall s, \tau_{exe}(j_p(s))$	[1, 2] ms	signal radius	[200, 600] m
$R$	500	$R'$	100000
$L$	10	$\epsilon$	$2 \times 10^{-4}$
$P$	10	it	300
$\mathbb{P}_m$	10%	$\mathbb{P}_c$	80%

*The microservices and candidates:* The number of microservices in the application  $Q$  is set as 10 in default. For each microservice  $q$ , the number of its candidates is uniformly sampled from the integer interval [2, 5]. In each replication  $\mathbf{W}^r$ , the service composition scheme of the  $i$ th end user is sampled according to  $\mathbb{P}(E(s(i)))$ . Considering that there is no commonly used dataset for microservice composition,  $\forall q \in \mathcal{Q}, \forall c \in \mathcal{C}_q$ , we generate  $\mathbb{P}(E(s_q^c))$  uniformly to avoid any bias.

*The pre-5G HetNet:* The experiment is conducted based on the geolocation information of base stations and end users within the Melbourne CBD area contained in the EUA dataset [17]. In our simulations, we choose 500 end users and 40 SBSs uniformly from the dataset in default. The coverage radius of each SBS is sampled from [200, 600] meters uniformly.  $\forall i \in \mathcal{N}, \forall j \in \mathcal{M}, \frac{1}{d(i,j)} = 1$  MHz. In addition, the maximum hops between any two SBSs can not larger than 4.  $\forall j \in \mathcal{M}, b_j$  is chosen from the integer interval  $[b_l, b_u]$ . We set  $b_l = 3, b_u = 5$  in default.

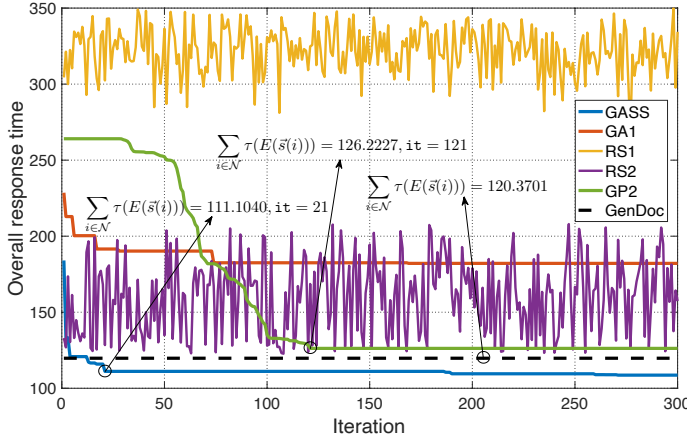
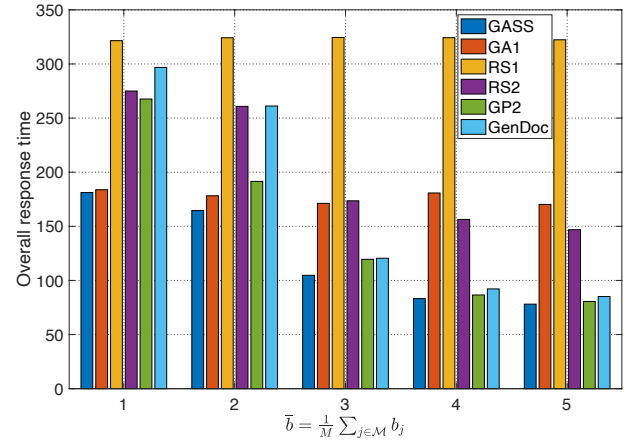
(a) The convergence rate of all the algorithms with  $N = 500$ ,  $M = 40$ .(b) The overall response time vs.  $\bar{b}$ .

Fig. 4. Algorithm performance comparison.

### 6.3 Experiment Results

The experiments are conducted to analyze the optimality and scalability of the proposed algorithm.

#### 6.3.1 Optimality

As shown in Fig. 4(a), GASS outperforms all the other algorithms in the overall response time, i.e.,  $\sum_{i \in N} \tau(E(s(i)))$ . Specifically, within 300 iterations, GASS outperforms GenDoc, GP2, RS2, GA1, and RS1 by 10.81%, 16.20%, 44.81%, 67.66%, and 196.43%, respectively. The result verifies both the rationality of redundant placement and the optimality of our algorithm. As for the former, it is verified by that all the algorithms implemented in scenario #2 perform better than the algorithms implemented in scenario #1. The superiority of redundant placement lies in that it takes full advantage of the distributed SBSs' limited resources. In this case, the processing of end users' service requests can surely be balanced. As for the latter, it is verified by that GASS can converge to an approximate optimal solution, i.e. 111.1040 ms at a very rapid rate. The solution achieved at the 21<sup>th</sup> iteration is already better than all the other algorithms.

In addition to the above phenomena, it is interesting to see that GP2 can obtain a relatively good result. We can verify that for each deployment, what GP2 adopts is the optimal operation. For each microservice, only the most frequently requested candidate has the privilege to be deployed, which ensures that the maximum number of mobile devices can enjoy their optimal situation. In comparison, GenDoc consists of *greedy* placement (server configuration) and *dynamic programming*-based microservice scheduling, which is not overbearing.

Fig. 4(b) shows that GASS can keep on top under different conditions. In this figure, the horizontal axis is the mean value of can-be-deployed candidates of each SBS  $j$ , i.e.  $b_j$ . We can find that for those algorithms which are implemented in scenario #1,  $\bar{b}$  has no significant effect on their performance. The reason is that whatever  $b_j$  takes, only one candidate can be deployed on each SBS  $j$ . It is also why GA1 can achieve a similar result with GASS when  $\bar{b} = 1$ . By contrast, with the increase in  $\bar{b}$ , all the algorithms implemented in scenario #2 enjoy less response

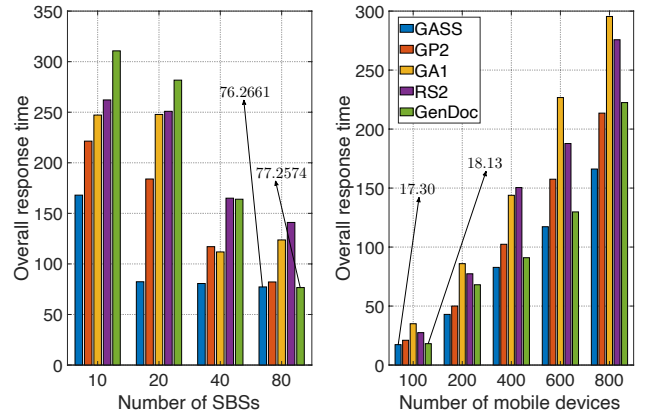


Fig. 5. Overall response time vs. scale of the HetNet.

time. The result is obvious because  $b_j$  decides the upper limit of resources, which is the key influence factor of time consumption. It is also worth noting that when  $\bar{b}$  increases, the gap between GASS, GP2, and GenDoc is likely to narrow. This is exactly the embodiment of the trade-off between algorithm improvement and resource promotion. When resources are rich, even poorly performing algorithms can produce good results.

#### 6.3.2 Scalability

Scalability is embodied in two aspects, the HetNet and the application (microservices and candidates).

**The HetNet:** the scale of the HetNet is embodied in two variables, the number of SBSs  $M$  and the number of mobile devices  $N$ . The left of Fig. 5 demonstrates the impact of  $M$  on the overall response time. Generally, as  $M$  increases, the response time decreases. This is because more SBSs can provide more resources, which greatly helps to realize *near-request processing*. Even so, the superiority of GASS is obvious, as it is always the best of five algorithms whatever  $M$  takes (RS1 is discarded). Thus, the scalability of GASS holds. Besides, there are some noteworthy phenomena. Firstly, the response time of GA1 has a slightly rising trend when  $M$  increases from 40 to 80. This is because when  $M$  increases, the

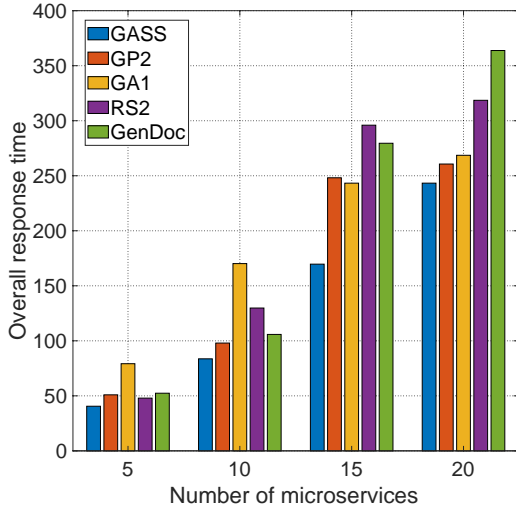


Fig. 6. Overall response time vs. number of the microservices.

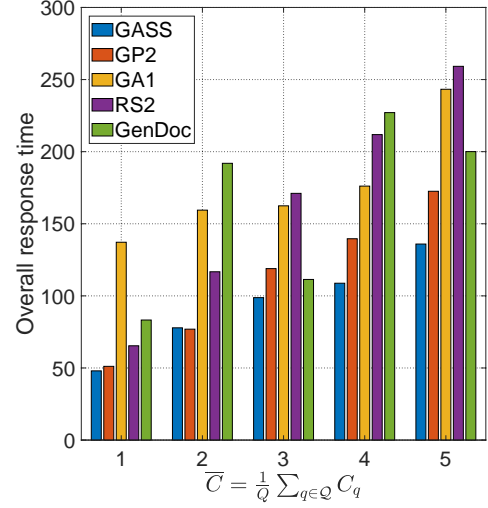
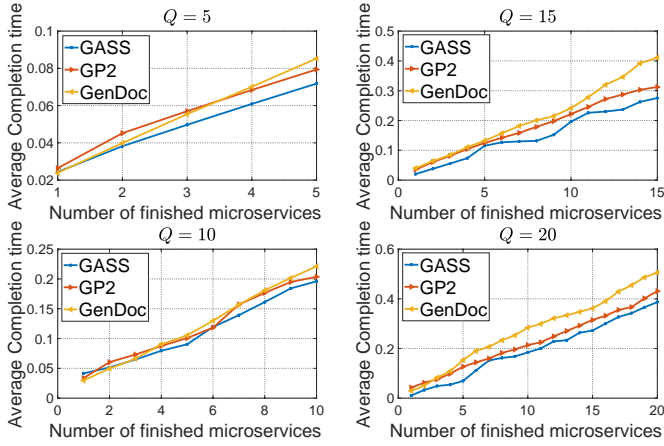
Fig. 8. Overall response time vs.  $\bar{C}$ .

Fig. 7. Average response time vs. number of the microservices.

dimension of feasible solution increases, which greatly expands the solution space. Meanwhile, the connected graph of SBSs become sparse, which leads to more hops to transfer data streams. Under this circumstances, 300 iterations might not be enough to achieve an optimal enough solution. However, GASS is not effected because the solution space of GASS is much smaller. The phenomenon verifies the second advantage displayed in Subsection 4.4. Secondly, when  $M$  increases, the gap between GASS, GP2, and GenDoc is likely to narrow. This phenomenon has been captured in Fig. 4(b). No matter increasing  $M$  or  $\bar{b}$ , the resources of SBSs are increasing, and poorly performing algorithms can produce good results.

The right of Fig. 5 demonstrates the impact of  $M$  on the overall response time. For all the implemented algorithms, the overall response time increases as  $N$  increases while the solution achieved by GASS is always the best. It is interesting that the gaps between those benchmark policies and GASS increase as  $N$  increases. It indicates that GASS is robust to the computation complexity of the fitness function. Thus, the scalability of GASS holds.

**The application:** the scale of the application is embodied in two variables, the number of microservices  $Q$

and the average number of candidates per microservice  $\bar{C} \triangleq \frac{1}{Q} \sum_{q \in Q} C_q$ . It can be concluded that GenDoc and GP2 are competitive while RS1, RS2 and GA1 are obviously lagging behind. Thus, in the following analysis, we only compare GASS with GenDoc and GP2 in terms of the average completion time.

Fig. 6 and Fig. 7 demonstrates the impact of the scale of microservices. From Fig. 6 we can find that GASS can keep on top whatever  $Q$  is. Correspondingly, Fig. 7 demonstrates the involution of the average completion time per mobile device when  $Q$  is 5, 10, 15, and 20, respectively. In all cases, GASS achieves the minimum average completion time no matter how many microservices have been finished. In our experiments, the maximum  $\mathbb{E}[\sum_{q \in Q} C_q]$  is 70 while  $\mathbb{E}[\sum_{j \in \mathcal{M}} b_j]$  is 160. Theoretically, if the expected number of all candidates  $\mathbb{E}[\sum_{q \in Q} C_q]$  does not exceed the expected number of all the can-be-deployed candidates  $\mathbb{E}[\sum_{j \in \mathcal{M}} b_j]$ , GASS can maintain its competitive edge. This is because no requests from end users need to be processed by cloud, and the time-consuming backbone can be saved. This advantage is not hold by the benchmark policies. Fig. 8 and Fig. 9 demonstrates the impact of the scale of candidates. Similarly, from Fig. 8, we can find that GASS outperforms GP2 and GenDoc in most cases. From Fig. 9 we can find that GASS always achieves the minimum average completion time when  $\bar{C}$  is 2, 3, 4, and 5.

Fig. 6 ~ Fig. 9 verifies that the scalability of GASS holds in terms of the number of microservices and candidates.

TABLE 3  
Impact of population size, crossover probability, and mutation probability.

$P$	$\sum_{i \in \mathcal{N}} \tau_i$	$\mathbb{P}_c$	$\sum_{i \in \mathcal{N}} \tau_i$	$\mathbb{P}_m$	$\sum_{i \in \mathcal{N}} \tau_i$
5	114.6762	20%	114.5866	20%	110.4808
10	109.9742	40%	112.2296	40%	105.8360
15	109.6814	60%	116.8252	60%	112.7771
20	113.4527	80%	111.4439	80%	109.2450
25	111.8812	100%	116.1299	100%	112.6478

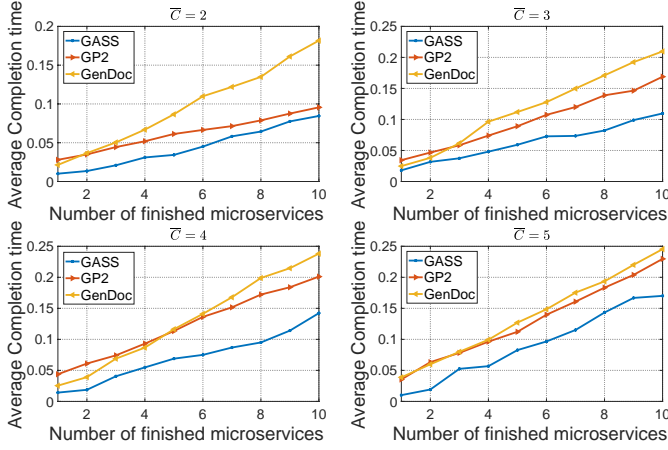


Fig. 9. Average response time vs.  $\bar{C}$ .

**This superparameters of GASS:** Table 3 demonstrates the overall response time of GASS under different population size, probability of mutation  $\mathbb{P}_m$ , and probability of crossover  $\mathbb{P}_c$ , respectively. We can find that their impact is minor on the optimality of GASS. As a result, no more detailed discussion is launched.

## 7 RELATED WORKS

Service computing based on traditional cloud data-centers has been extensively studied in the last several years, especially service selection for composition [24] [25] [26], service provision [27], discovery [28], and so on. However, putting *everything about services* onto the distributed and heterogenous edge is still an area waiting for exploration. Multi-access Edge computing, as a increasingly popular computation paradigm, is facing the transition from theory to practice. The key to the transition is the placement and deployment of service instances.

In the last two years, service placement at the distributed edge has been tentatively explored from the perspective of Quality of Experience (QoE) of end users or the budget of ASPs. For example, Ouyang et al. study the problem in a mobility-aware and dynamic way [6]. Their goal is to optimize the long-term time averaged migration cost triggered by *user mobility*. They develop two efficient heuristic schemes based on the Markov approximation and best response update techniques to approach a near-optimal solution. System stability is also guaranteed by Laypunov optimization technique. Chen et al. study the problem in a spatio-temporal way, under a limited budget of ASPs [9]. They pose the problem as a combinatorial contextual bandit learning problem and utilize Multi-armed Bandit (MAB) theory to *learn* the optimal placement policy. However, the proposed algorithm is time-consuming and faces the curse of dimensionality. Except for the typical examples listed above, there also exist works dedicated on joint resource allocation and load balancing in service placement [7] [8] [10] [29]. However, as we have mentioned before, those works only study the to-be-placed services in an atomic way. The correlated and composite property of services is not taken into consideration. Besides, those works do not tell us how to apply their algorithms to the service deployment in

a practical system. To address these deficiencies, we navigate the service placement and deployment from the view of production practices. Specifically, we adopt redundant placement to the correlated microservices, which can be unified managed by Kubernetes.

The idea of redundancy has been studied in parallel-server systems and computing clusters [30] [31] [32]. The basic idea of redundancy is dispatching the same job to multiple servers. The job is considered done as soon as it completes service on any one server [15]. Typical job redundancy model is the  $S\&X$  model, where  $X$  is the job's inherent size, and  $S$  is the server slowdown. It is designed based on the weakness of the traditional Independent Runtimes (IR) model, where a jobs replicas experience independent runtimes across servers. Unfortunately, although the  $S\&X$  model indeed captures the practical characteristics of real systems, it still face great challenges to put it into use in service deployment at the edge because the geographically distribution and heterogeneity of edge sites are not considered. To solve the problem, in this paper we redesign the entire model while the idea of redundancy is kept.

This work significantly extends our preliminary work [33]. To improve the practicability, we analyze the response time of each mobile device in a more rigorous manner, and improve it by always finding the nearest available edge site. We also take the uncertainty of end users' service composition scheme into consideration. It greatly increases the complexity but is of signality. Most important of all, we embed the idea of redundancy into the problem and design an algorithm with a faster convergence rate.

## 8 CONCLUSION

In this paper, we study a redundant placement policy for the deployment of microservice-based applications at the distributed edge. We first demonstrate the typical HetNet in the near future, and then explore the possibilities of the deployment of composite microservices with containers and Kubernetes. After that, we model the redundant placement as a stochastic optimization problem. For the application with composite and correlated microservices, we design the SAA-based framework SAA-RP and the GASS algorithm to dispatch microservice instances into edge sites. By creating multiple access to services, our policy boosts a faster response for mobile devices significantly. SAA-RP not only take the uncertainty of microservice composition schemes of end users, but also the heterogeneity of edge sites into consideration. The experimental results based on a real-world dataset show both the optimality of redundant placement and the efficiency of GASS. In addition, we give guidance on the implementation of SAA-RP with Kubernetes. In our future work, we will hammer at the implementation of the redundant deployment of complex DAGs with arbitrary shape.

## ACKNOWLEDGMENTS

This research was partially supported by the National Key Research and Development Program of China (No. 2017YFB1400601), Key Research and Development Project of Zhejiang Province (No. 2017C01015), National Science



Foundation of China (No. 61772461), Natural Science Foundation of Zhejiang Province (No. LR18F020003 and No. LY17F020014).

## REFERENCES

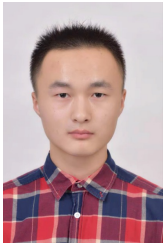
- [1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, Fourthquarter 2017.
- [2] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1657–1681, thirdquarter 2017.
- [3] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet of Things Journal*, pp. 1–1, 2020.
- [4] "Docker: Modernize your applications, accelerate innovation." [Online]. Available: <https://www.docker.com/>
- [5] "Kubernetes: Production-grade container orchestration." [Online]. Available: <https://kubernetes.io/>
- [6] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2333–2345, Oct 2018.
- [7] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein, "It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, July 2018, pp. 365–375.
- [8] B. Gao, Z. Zhou, F. Liu, and F. Xu, "Winning at the starting line: Joint network selection and service placement for mobile edge computing," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, April 2019, pp. 1459–1467.
- [9] L. Chen, J. Xu, S. Ren, and P. Zhou, "Spatiotemporal edge service placement: A bandit learning approach," *IEEE Transactions on Wireless Communications*, vol. 17, no. 12, pp. 8388–8401, Dec 2018.
- [10] F. Ait Salait, F. Desprez, A. Lebre, C. Prud'homme, and M. Abderrahim, "Service placement in fog computing using constraint programming," in *2019 IEEE International Conference on Services Computing (SCC)*, July 2019, pp. 19–27.
- [11] Y. Chen, S. Deng, H. Zhao, Q. He, Y. Li, and H. Gao, "Data-intensive application deployment at edge: A deep reinforcement learning approach," in *2019 IEEE International Conference on Web Services (ICWS)*, July 2019, pp. 355–359.
- [12] L. Liu, H. Tan, S. H.-C. Jiang, Z. Han, X.-Y. Li, and H. Huang, "Dependent task placement and scheduling with function configuration in edge computing," in *Proceedings of the International Symposium on Quality of Service*, ser. IWQoS 19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: <https://doi.org/10.1145/3326285.3329055>
- [13] L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "Deploying microservice based applications with kubernetes: Experiments and lessons learned," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, July 2018, pp. 970–973.
- [14] L. A. Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "Kubernetes as an availability manager for microservice applications," *CoRR*, vol. abs/1901.04946, 2019.
- [15] K. Gardner, M. Harchol-Balter, A. Scheller-Wolf, and B. Van Houdt, "A better model for job redundancy: Decoupling server slowdown and job size," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3353–3367, Dec 2017.
- [16] "Kubernetes scheduler." [Online]. Available: <https://kubernetes.io/docs/concepts/scheduling/kube-scheduler/>
- [17] P. Lai, Q. He, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *Service-Oriented Computing*. Cham: Springer International Publishing, 2018, pp. 230–245.
- [18] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, "Network flows," 1988.
- [19] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, "Network slicing in 5g: Survey and challenges," *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.
- [20] A. J. Kleywegt, A. Shapiro, and T. Homem-de Mello, "The sample average approximation method for stochastic discrete optimization," *SIAM Journal on Optimization*, vol. 12, no. 2, pp. 479–502, 2002.
- [21] C. Robert and G. Casella, *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.
- [22] E. K. Burke, G. Kendall et al., *Search methodologies*. Springer, 2005.
- [23] B. L. Miller, D. E. Goldberg et al., "Genetic algorithms, tournament selection, and the effects of noise," *Complex systems*, vol. 9, no. 3, pp. 193–212, 1995.
- [24] S. Deng, H. Wu, W. Tan, Z. Xiang, and Z. Wu, "Mobile service selection for composition: An energy consumption perspective," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 3, pp. 1478–1490, July 2017.
- [25] H. Yuan, J. Bi, and M. Zhou, "Temporal task scheduling of multiple delay-constrained applications in green hybrid cloud," *IEEE Transactions on Services Computing*, pp. 1–1, 2018.
- [26] Q. Wu, M. Zhou, Q. Zhu, and Y. Xia, "Vcg auction-based dynamic pricing for multigranularity service composition," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, pp. 796–805, 2018.
- [27] H. Wu, S. Deng, W. Li, J. Yin, Q. Yang, Z. Wu, and A. Y. Zomaya, "Revenue-driven service provisioning for resource sharing in mobile cloud computing," in *Service-Oriented Computing*, 2017, pp. 625–640.
- [28] W. Chen, I. Paik, and P. C. K. Hung, "Constructing a global social service network for better quality of web service discovery," *IEEE Transactions on Services Computing*, vol. 8, no. 2, pp. 284–298, March 2015.
- [29] Q. Fan and N. Ansari, "On cost aware cloudlet placement for mobile edge computing," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 4, pp. 926–937, 2019.
- [30] H. Deng, T. Zhao, and I. Hou, "Online routing and scheduling with capacity redundancy for timely delivery guarantees in multi-hop networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1258–1271, June 2019.
- [31] A. abdi, A. Girault, and H. R. Zarandi, "Erpot: A quad-criteria scheduling heuristic to optimize execution time, reliability, power consumption and temperature in multicores," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2193–2210, Oct 2019.
- [32] H. Xu, G. De Veciana, W. C. Lau, and K. Zhou, "Online job scheduling with redundancy and opportunistic checkpointing: A speedup-function-based analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 897–909, April 2019.
- [33] Y. Chen, S. Deng, H. Ma, and J. Yin, "Deploying data-intensive applications with multiple services components on edge," *Mobile Networks and Applications*, Apr 2019.



**Hailiang Zhao** received the B.S. degree in 2019 from the school of computer science and technology, Wuhan University of Technology, Wuhan, China. He is currently pursuing the Ph.D. degree with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. He has been a recipient of the Best Student Paper Award of IEEE ICWS 2019. His research interests include edge computing, service computing and machine learning.



**Shuiguang Deng** is currently a full professor at the College of Computer Science and Technology in Zhejiang University, China, where he received a BS and PhD degree both in Computer Science in 2002 and 2007, respectively. He previously worked at the Massachusetts Institute of Technology in 2014 and Stanford University in 2015 as a visiting scholar. His research interests include Edge Computing, Service Computing, Mobile Computing, and Business Process Management. He serves as the associate editor for the journal IEEE Access and IET Cyber-Physical Systems: Theory & Applications. Up to now, he has published more than 100 papers in journals and refereed conferences. In 2018, he was granted the Rising Star Award by IEEE TCSVC. He is a fellow of IET and a senior member of IEEE.



**Zijie Liu** received the B.S. degree in 2018 from the school of computer science and technology, Huazhong University of Science and Technology, Wuhan, China. He is now pursuing the master degree with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. His research interests include edge computing and software engineering.



**Jianwei Yin** received the Ph.D. degree in computer science from Zhejiang University (ZJU) in 2001. He was a Visiting Scholar with the Georgia Institute of Technology. He is currently a Full Professor with the College of Computer Science, ZJU. Up to now, he has published more than 100 papers in top international journals and conferences. His current research interests include service computing and business process management. He is an Associate Editor of the IEEE Transactions on Services Computing.



**Schahram Dustdar** is a Full Professor of Computer Science (Informatics) with a focus on Internet Technologies heading the Distributed Systems Group at the TU Wien. He is 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 at the University of Groningen (RuG), The Netherlands.

From December 2016 until January 2017 he was a Visiting Professor at the University of Sevilla, Spain and from January until June 2017 he was a Visiting Professor at UC Berkeley, USA. 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 the IBM Faculty Award (2012). He is an Associate Editor of IEEE Transactions on Services Computing, ACM Transactions on the Web, and ACM Transactions on Internet Technology and on the editorial board of IEEE Internet Computing. He is the Editor-in-Chief of Computing (an SCI-ranked journal of Springer).