Distributed Redundancy Scheduling 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 deployment 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 redundancy scheduling algorithm, named SAA-RS, 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 scheduled and with how many instances as well as on which edge sites to deploy them. Benchmarks are implemented in two scenarios, where redundancy scheduling is allowed and not, respectively. Numerical results based on a real-world dataset verifies that SAA-RS significantly outperforms four benchmarks by 60.89%, 71.51%, 79.23%, and 84.91%, respectively.

Index Terms—Redundancy Scheduling, Multi-access Edge Computing, Service Deployment, 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].

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 in cellular tower sites, there are exciting possibilities that microservice-based applications can be delivered to mobile devices with a unified service provision platform. Container technologies, represented by Docker [3], and its dominant orchestration and maintenance tool, Kubernetes [4], 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. It is naturally suitable for building *edge-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 network.

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 [5] [6] [7] [8] [9] [10]. However, those works often 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 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 slowdown 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 failured ones, 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 [11] [12]. Therefore, with the vanilla version of Kubernetes, high availability might not be ensured, especially for the distributed edge.

In order to solve the above problems, we propose a distributed redundancy scheduling algorithm, i.e., Sample Av-

- *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 Universitt Wien, 1040 Vienna, Austria. E-mail: dustdar@dsg.tuwien.ac.at*
- *Shuiguang Deng is the corresponding author.*

erage Approximation-based Greedy Redundancy Scheduling (SAA-RS), for a microservice-based applications with sequential combinatorial structure. For this application, if all of the microservices 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-RS, which allows that one microservice to be dispatched to multiple edge sites. By creating multiple microservice instances, it boosts a faster response to service requests. To be specific, it alleviates the risk of a long delay incurred when a microservice is assigned to only one edge site. With one microservice instance deployed on more than one edge site, requests from different end users at different locations can be balanced, so as to ensure the 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 [13]. However, which kind of microservice requires redundancy and how many instances should be deployed cannot be decided if out of 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 microservice 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 [13].

As a result, we do not release the microservice instances but periodically update them based on the observations of service demand status during that period. Specifically, we derive expressions to decide each microservice should be scheduled with how many instances and which edge sites to deploy them. By collecting user requests for different service composition schemes, we model the distributed redundancy scheduling 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 greedy policy. Performance analysis and numerical results are provided to verify its practicability. Our main contributions are as follows.

1) A distributed redundancy scheduling algorithm at the edge, i.e. SAA-RS, for microservice-based applications is studied. SAA-RS can decide each microservice should be scheduled with how many instances and which edge sites to deploy them. It can be put into practice by superseding the default scheduler of Kubernetes, i.e. `kube-scheduler` [14].
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 a real-world dataset. We also provide the performance analysis on algorithm optimality and convergence rate. The numerical results verify the practicability and superiority of our algorithm, compraed with several typical benchmarks.

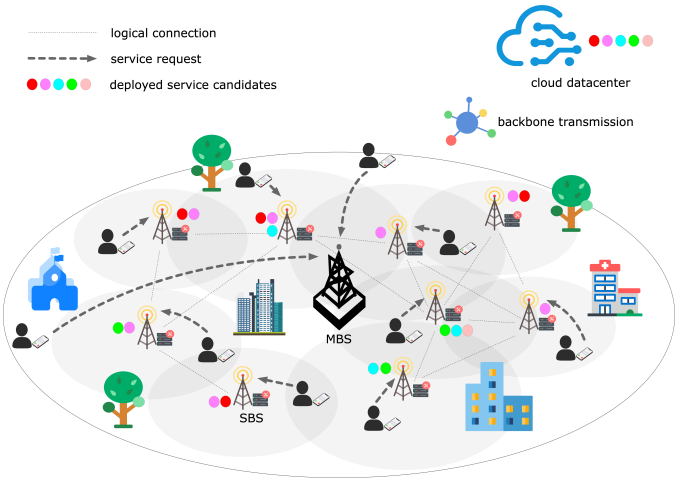


Fig. 1. A typical scenario for a pre-5G HetNet.

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-RS algorithm 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 scheduling models. Section 8 concludes this paper.

2 Motivation Scenarios

2.1 The Heterogeneous Network

Let us consider a typical scenario for a pre-5G Heterogeneous Network (HetNet), which is the physical foundation of service placement and redundancy scheduling 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 through X2 links 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)[1], 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

1. Whether SBSs are logically connected comes down to their IP segements.

small-scale datacenter attached for the deployment of microservices and the scheduling 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 Redundancy Scheduling for Microservices

A microservice-based application is consist of multiple microservices. Each microservice can be performed 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[2]. Finally, 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[3], only Alipay is supported for online payment. Without loss of generality, this paper only care about a *sequentially* composed application.

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, mobile devices send their service requests to the nearest SBS for the strongest signal of the established link. However, if there has no accessible SBS around, the requests have to be responsed by the MBS and processed by cloud datacenters. Therefore, the response status of each candidate can be divided into three circumstances:

1) The requested candidate is deployed at the chosen SBS. It will be processed by the SBS instantly.
2) The requested candidate is not depolyed at the chosen SBS but accessible on other SBSs, which leads to multi-hop transfers between the SBSs until the request is responded by another SBS.
3) The requested candidate is not deployed on any SBSs in the access network. It can only be processed by cloud through backbone transmission.

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

1) The previous microservice is processed by an SBS. Under this circumstance, for this microservice, if its instance can be found within the HetNet, multi-hop transfers might be required. Otherwise, it has to be processed by cloud.
2) The previous microservice is processed by cloud. Under this circumstance, the subsequent microservices should always be responsed by cloud without unnecessary backhual.

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

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

TABLE 1
Summary of key notations.

Notation	Description
M	the number of SBSs, $M = \|\mathcal{M}\|$
N	the number of mobile device, $N = \|\mathcal{N}\|$
Q	the number of microservices in the application, $Q = \|\mathcal{Q}\|$
$t_q, q \in \mathcal{Q}$	the qth microservice in the application
$\mathcal{M}_i$	the set of SBSs that can be connected by the ith mobile device
$\mathcal{N}_j$	the set of mobile devices that can be connected by the jth SBS
C_q	the number of candidates of the qth microservice, $C_q = \|\mathcal{C}_q\|, q \in \mathcal{Q}$
$s^c(t_q), c \in \mathcal{C}_q$	the cth candidate of the qth microservice
$\mathcal{D}(s^c(t_q))$	the set of SBSs on which $s^c(t_q)$ is deployed
$E(s^c(t_q))$	the random event that for microservice t_q, the cth candidate is selected for execution
$j^\star(s^{c_1^i}(t_1))$	the nearest SBS to the ith mobile device
$j_p(s)$	the SBS who actually runs the candidate s
$d(i,j)$	the Euclidean distance between the ith mobile device and the jth SBS
$\tau_{in}(s)$	the data uplink transmission time of the candidate s
$\tau_{exe}(j_p(s))$	the execution time of the candidate s on the j_pth SBS
$\zeta(j_1, j_2)$	the hop-count between the j_1th SBS and the j_2th SBS
b_j	the maximum number of microservice instances can be deployed on the jth SBS

Our job is to find an optimal redundancy scheduling 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.

3 System Model

The HetNet is consist of N mobile devices, indexed by $\mathcal{N} \triangleq \{1, ..., i, ..., N\}$, M SBSs, indexed by $\mathcal{M} \triangleq \{1, ..., j, ..., M\}$, 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 ith mobile devices. Correspondingly, $\mathcal{N}_j$ denotes the set of mobile devices that are covered by the jth 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 $\langle t_1, ..., t_Q \rangle$, indexed by $\mathcal{Q}$. $\forall q \in \mathcal{Q}$, microservice t_q has C_q candidates $\boldsymbol{s}(t_q) \triangleq \{s^1(t_q), ..., s^c(t_q), ..., s^{C_q}(t_q)\}$, indexed by $\mathcal{C}_q$. Let us use $\mathcal{D}(s^c(t_q)) \subseteq \mathcal{M}$ to denote the set of SBSs on which $s^c(t_q)$ is deployed. Our redundancy scheduling policy allows that $|\mathcal{D}(s^c(t_q))| > 1$, which means one microservice's instance could be dispatched to more than one SBS. Besides, let us use $E(s^c(t_q)), c \in \mathcal{C}_q$ to represent

that for microservice t_q, the cth candidate is selected for execution. Without loss of generality, $E(s^c(t_q))$ can be viewed as a random event with an unknown distribution. Further, $\mathbb{P}(E(s^c(t_q)))$ denote the probability that $E(s^c(t_q))$ happens. Thus, for each mobile device, his selected candidates for the composite service can be described as a Q-tuple:

$$\boldsymbol{E}(s^{c_q}(t_q))_{q=1}^{Q} \triangleq \langle E(s^{c_1}(t_1)), ..., E(s^{c_Q}(t_Q))\rangle, \quad (1)$$

where $q \in \mathcal{Q}, c_q \in \mathcal{C}_q$.

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

Definition 3.1. **Correlations of Composite Service** $\forall q \in \mathcal{Q}\backslash\{1\}$, $c_1 \in \mathcal{C}_{q-1}$, and $c_2 \in \mathcal{C}_q$, candidate $s^{c_2}(t_q)$ and $s^{c_1}(t_{q-1})$ are correlated *iff* $\mathbb{P}(E(s^{c_2}(t_q))|E(s^{c_1}(t_{q-1}))) \equiv 1$, and $\forall c_2' \in \mathcal{C}_q\backslash\{c_2\}$, $\mathbb{P}(E(s^{c_2'}(t_q))|E(s^{c_1}(t_{q-1}))) \equiv 0$.

With the above definition, the probability $\mathbb{P}(\boldsymbol{E}(s^{c_q}(t_q))_{q=1}^{Q})$ can be calculated by

$$\mathbb{P}(\boldsymbol{E}(s^{c_q}(t_q))_{q=1}^{Q}) = \mathbb{P}(E(s^{c_1}(t_1))) \cdot \prod_{q=2}^{Q} \mathbb{P}(E(s^{c_q}(t_q))|E(s^{c_{q-1}}(t_{q-1}))). \quad (2)$$

3.2 Calculating the Respone Time

The response time of one microservice is consist 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 microservice. If all requests are responsed within the access network, most of the time is spent on routing with multi-hops between SBSs. Notice that except the last one, each microservice's data uplink transmission time is equal to the data downlink transmission time of its previous microservice.

3.2.1 For the Initial Microservice

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

1) In the first case, if the candidate $s^{c_1^i}(t_1)$ is not deployed on any SBSs from $\mathcal{M}$, i.e. $\mathcal{D}(s^{c_1^i}(t_1)) = \varnothing$, the request still has to be forwarded to cloud datacenter through backbone transmission.
2) In the second case, when $\mathcal{D}(s^{c_1^i}(t_1)) \neq \varnothing$ and $j^\star(s^{c_1^i}(t_1)) \in \mathcal{D}(s^{c_1^i}(t_1))$, the request can be directly processed by SBS $j^\star(s^{c_1^i}(t_1))$ without any hops within the access network.
3) In the third case, $\mathcal{D}(s^{c_1^i}(t_1)) \neq \varnothing$ and $j^\star(s^{c_1^i}(t_1)) \notin \mathcal{D}(s^{c_1^i}(t_1))$, the request has to be responsed by $j^\star(s^{c_1^i}(t_1))$ and processed by another SBS from $\mathcal{D}(s^{c_1^i}(t_1))$.

Without loss of generality, the expenditure of time on wireless access is directly proportional to the distance from mobile device i to the connected base station j, which is denoted by $d(i,j)$. Besides, whatever communication mechanism is adopted, the expenditure of time on routing is directly proportional to the number of hops between the source node and destination node. Let us denote the shortest number of hops from node j_1 to node j_2 in the connected graph by $\zeta(j_1, j_2)$. We use $j_p(s)$ to denote the SBS who actually runs the sth service candidate. For $q = 1$, the calculation of $j_p(s^{c_q^i}(t_q))$ is summarized in (4). $j_p(s) =$ cloud means that the candidate instance is running on cloud. In conclusion, the data uplink transmmision time $\tau_{in}(s^{c_1^i}(t_1))$ is summarized in (3), where 0 is the index of the MBS, τ_b is the time on backbone transmission, α is a positive constant for wireless access, and β is a positive constant representing the rate of X2 link. Without loss of generality, $\forall i \in \mathcal{N}, j, j_1, j_2 \in \mathcal{M}_i$, $\tau_b \gg \alpha \cdot d(i,j) \approx \beta \cdot \zeta(j_1, j_2)$. We use $\tau_{exe}(j_p(s^{c_1^i}(t_1)))$ to denote the microservice execution time on the SBS j_p for the candidate $s^{c_1^i}(t_1)$. Compared with the complicated routing among SBSs, the difference on execution time can be omitted. The data downlink transmission time is the same as the uplink time of the next microservice, which is discussed below.

3.2.2 For the Intermediate Microservices

For the ith mobile device and its selected candidate $s^{c_q^i}(t_q)$ of microservice t_q, where $q \in \mathcal{Q}\backslash\{1, Q\}$, $c_q^i \in \mathcal{C}_q$, the analysis of its data uplink transmission time is correlated with $j_p(s^{c_{q-1}^i}(t_{q-1}))$, i.e. the SBS who run the candidate $s^{c_{q-1}^i}(t_{q-1})$. $\forall q \in \mathcal{Q}\backslash\{1\}$, the calculation of $j_p(s^{c_q^i}(t_q))$ is summarized in (5). (**I**) If $j_p(s^{c_{q-1}^i}(t_{q-1})) \in \mathcal{D}(s^{c_q^i}(t_q))$, then the data downlink transmission time of previous microservice t_{q-1}, which is also the data uplink transmission time of current microservice, t_q, is zero, i.e., $\tau_{out}(s^{c_{q-1}^i}(t_{q-1})) = \tau_{in}(s^{c_q^i}(t_q)) = 0$. It is because both $s^{c_{q-1}^i}(t_{q-1})$ and $s^{c_q^i}(t_q)$ are deployed on the SBS $j_p(s^{c_{q-1}^i}(t_{q-1}))$, which leads to the number of hops being zero. (**II**) If $j_p(s^{c_{q-1}^i}(t_{q-1})) \notin \mathcal{D}(s^{c_q^i}(t_q))$, a classified discussion has to be launched.

1) In the first case, $j_p(s^{c_{q-1}^i}(t_{q-1})) =$ cloud, which means the request for t_{q-1} from the ith mobile device is responded by cloud datacenter. In this case, the invocation for t_q can directly processed by cloud without backhual. Thus, the data uplink transmission time of t_q is zero.
2) In the second case, $j_p(s^{c_{q-1}^i}(t_{q-1})) \neq 0$. However, $\mathcal{D}(s^{c_q^i}(t_q)) = \varnothing$. Under this circumstance, the candidates of microservice t_q is not deployed on any SBSs from $\mathcal{M}$. As a result, the invocation for t_q has to be responded by cloud datacenter through backbone transmission.
3) In the third case, $j_p(s^{c_{q-1}^i}(t_{q-1})) \neq 0$, $\mathcal{D}(s^{c_q^i}(t_q)) \neq \varnothing$ but $\{j_p(s^{c_{q-1}^i}(t_{q-1}))\} \bigcap \mathcal{D}(s^{c_q^i}(t_q)) = \varnothing$, which means both $s^{c_{q-1}^i}(t_{q-1})$ and $s^{c_q^i}(t_q)$ can be processed by the SBSs from $\mathcal{M}$ but not the same one. In this case, we can calculate the data uplink transmission time by finding the shortest path from $j_p(s^{c_{q-1}^i}(t_{q-1}))$ to the SBSs in $\mathcal{D}(s^{c_q^i}(t_q))$.

$$\tau_{in}(s^{c_1^i}(t_1)) = \begin{cases} \alpha \cdot d(i,0) + \tau_b, & \mathcal{M}_i = \varnothing; \\ \alpha \cdot d(i, j^\star(s^{c_1^i}(t_1))) + \tau_b, & \mathcal{M}_i \neq \varnothing, \mathcal{D}(s^{c_1^i}(t_1)) = \varnothing; \\ \alpha \cdot d(i, j^\star(s^{c_1^i}(t_1))), & \mathcal{M}_i \neq \varnothing, j^\star(s^{c_1^i}(t_1)) \in \mathcal{D}(s^{c_1^i}(t_1)); \\ \alpha \cdot d(i, j^\star(s^{c_1^i}(t_1))) + \beta \cdot \min_{j^\bullet \in \mathcal{D}(s^{c_1^i}(t_1))} \zeta(j^\star(s^{c_1^i}(t_1)), j^\bullet), & \text{otherwise} \end{cases} \tag{3}$$

$$j_p(s^{c_1^i}(t_1)) = \begin{cases} \text{cloud}, & \mathcal{M}_i = \varnothing \text{ or } \mathcal{D}(s^{c_1^i}(t_1)) = \varnothing; \\ j^\star(s^{c_1^i}(t_1)), & \mathcal{D}(s^{c_1^i}(t_1)) \neq \varnothing, j^\star(s^{c_1^i}(t_1)) \in \mathcal{D}(s^{c_1^i}(t_1)); \\ \operatorname{argmin}_{j^\bullet \in \mathcal{D}(s^{c_1^i}(t_1))} \zeta(j^\star(s^{c_1^i}(t_1)), j^\bullet), & \text{otherwise} \end{cases} \tag{4}$$

$$j_p(s^{c_q^i}(t_q)) = \begin{cases} \text{cloud}, & \mathcal{M}_i = \varnothing \text{ or } \mathcal{D}(s^{c_q^i}(t_q)) = \varnothing; \\ j_p(s^{c_{q-1}^i}(t_{q-1})), & \mathcal{D}(s^{c_q^i}(t_q)) \neq \varnothing, j_p(s^{c_{q-1}^i}(t_{q-1})) \in \mathcal{D}(s^{c_q^i}(t_q)); \\ \operatorname{argmin}_{j^\bullet \in \mathcal{D}(s^{c_q^i}(t_q))} \zeta(j_p(s^{c_{q-1}^i}(t_{q-1})), j^\bullet), & \text{otherwise} \end{cases} \tag{5}$$

$$\tau_{in}(s^{c_q^i}(t_q)) = \begin{cases} 0, & j_p(s^{c_{q-1}^i}(t_{q-1})) \in \mathcal{D}(s^{c_q^i}(t_q)) \text{ or } j_p(s^{c_{q-1}^i}(t_{q-1})) = \text{cloud}; \\ \tau_b, & j_p(s^{c_{q-1}^i}(t_{q-1})) \neq 0, \mathcal{D}(s^{c_q^i}(t_q)) = \varnothing; \\ \beta \cdot \min_{j^\bullet \in \mathcal{D}(s^{c_q^i}(t_q))} \zeta(j_p(s^{c_{q-1}^i}(t_{q-1}))), j^\bullet), & \text{otherwise} \end{cases} \tag{6}$$

$$\tau_{out}(s^{c_Q^i}(t_Q)) = \begin{cases} \tau_b + \alpha \cdot d(i,0), & j_p(s^{c_Q^i}(t_Q)) = 0; \\ \beta \cdot \zeta(j_p(s^{c_Q^i}(t_Q)), j^\star(s^{c_1^i}(t_1))) + \alpha \cdot d(i, j^\star(s^{c_1^i}(t_1))), & \text{otherwise} \end{cases} \tag{7}$$

$\forall q \in \mathcal{Q}\backslash\{1, Q\}$, the data uplink transmission time of microservice t_q, i.e. $\tau_{in}(s^{c_q^i}(t_q))$, which is also the data downlink transmission time of the previous microservice t_{q-1}, is summarized by (6).

3.2.3 For the Last Microservice

For the ith mobile device and its selected candidate $s^{c_Q^i}(t_Q)$ of the last microservice t_Q, the data uplink transmission time $\tau_{in}(s^{c_Q^i}(t_Q))$ can also be calculated by (6), with every q replaced by Q. However, for the data downlink transmission time $\tau_{out}(s^{c_Q^i}(t_Q))$, a classified discussion is required. (**I**) If $j_p(s^{c_Q^i}(t_Q)) = 0$, which means the chosen candidate of the last microservice is processed by cloud datacenter, the processed result need to be returned from cloud to the ith mobile device through backhual transmission[4]. (**II**) If $j_p(s^{c_Q^i}(t_Q)) \neq 0$, which means the chosen candidate of the last microservice is processed by a SBS from $\mathcal{M}$. In this case, the result is delivered from $j_p(s^{c_Q^i}(t_Q))$ to $j^\star(s^{c_1^i}(t_1))$ within the access network. (7) summarizes the calculation of $\tau_{out}(s^{c_Q^i}(t_Q))$. Recall that $j^\star(s^{c_1^i}(t_1))$ is the SBS who is direcly connected with mobile device i.

Based on the above analysis, the response time of the ith mobile device is

$$\tau(\boldsymbol{E}(s^{c_q^i}(t_q))_{q=1}^Q) = \sum_{q=1}^{Q} \Big(\tau_{in}(s^{c_q^i}(t_q)) + \tau_{exe}\big(j_p(s^{c_q^i}(t_q))\big)\Big) + \tau_{out}(s^{c_Q^i}(t_Q)). \tag{8}$$

4. In order to simplify the problem, the backhual can only be transferred by the MBS.

3.3 Problem Formulation

Our job is to find an optimal redundancy scheduling policy to minimize the overall latency under the limited memory and storage of SBSs. In order to *stylize* the problem formulation, let us use $\boldsymbol{W}(i) \triangleq (\texttt{canIdx}(t_1), ..., \texttt{canIdx}(t_Q))$ to denote the random vector on the chosen service composition scheme of the ith mobile device, where $\texttt{canIdx}(t_q)$ returns the index of the chosen candidate of the qth microservice. $\boldsymbol{W}(i)$ and $\boldsymbol{E}(s^{c_q^i}(t_q))_{q=1}^Q$ describe the same thing from different perspectives. However, the former is more concise. Then, we use $\boldsymbol{W} \triangleq (\boldsymbol{W}(1), ..., \boldsymbol{W}(N))$ to denote the global random vector by taking all mobile devices into account. The heterogeneity of edge sites is directly embodied in the number of *can-be-deployed* microservices. Let us use b_j to denote this number. For different SBS, b_j could vary considerably. The following constraint has to be satisified:

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

where $\mathbb{1}\{\cdot\}$ is the indicator function.

Let us use $\boldsymbol{x} \triangleq [\boldsymbol{x}(b_1), ..., \boldsymbol{x}(b_M)]^\top \in \mathcal{X}$ to denote the global *deploy-or-not* vector. $\forall j \in \mathcal{M}, \boldsymbol{x}(b_j)$ is a deployment vector for SBS j, whose length is b_j. Each element of $\boldsymbol{x}(b_j)$ is chosen from $\{0, 1, ..., \sum_{q=1} C_q\}$, i.e. the global index of each candidate. $\boldsymbol{x}(b_j) = 0$ means that this position does not choose any candidate. Notice that each candidate only have to be deployed once on a SBS. Thus, the number of possible values of $\boldsymbol{x}(b_j)$ 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 $\mathcal{X}$ is $\prod_{j \in \mathcal{M}} \big(b_j(\sum_{q \in \mathcal{Q}} C_q + 1) - \frac{b_j}{2}(b_j - 1)\big)$. It follows from the fact that the kth element of $\boldsymbol{x}(b_j)$ has $\sum_{q \in \mathcal{Q}} C_q + 2 - k$ choices. $\boldsymbol{x}$ and $\mathcal{D}(s^{c_q}(t_q))$ describe the same thing from different perspectives. But

the latter is not formalized. As a result, we can reconstitute $\tau(\boldsymbol{E}(s^{c_q^i}(t_q))_{q=1}^{Q})$ as $\tau(\boldsymbol{x}, \boldsymbol{W}(i))$. The optimization goal takes time consumption of all mobile devices into consideration, which looks like

$$g(\boldsymbol{x}) \triangleq \mathbb{E}[G(\boldsymbol{x}, \boldsymbol{W})] = \mathbb{E}[\sum_{i=1}^{N} \tau(\boldsymbol{x}, \boldsymbol{W}(i))]. \tag{10}$$

Thus, the redundancy scheduling problem can be written as

$$\begin{aligned} \mathcal{P}_1: \quad & \min_{\boldsymbol{x}\in\mathcal{X}} g(\boldsymbol{x}) \\ s.t. \quad & (9). \end{aligned}$$

3.4 Problem Analysis

$\mathcal{P}_1$ is a stochastic discrete optimization problem with independent variable $\boldsymbol{x} \in \mathcal{X}$, where $\mathcal{X}$ is a finite set. Besides, the problem has an uncertain random vector $\boldsymbol{W}$ with probability distribution $\mathbb{P}(\boldsymbol{E}(s^{c_q}(t_q))_{q=1}^{Q})$. The set $\mathcal{X}$ of feasible solutions, although finite, is very large. Therefore, enumeration approach is inadvisable.

Let us take a closer look at $\mathcal{P}_1$. Firstly, the random vector $\boldsymbol{W}$ is *exogenous* because the decision on $\boldsymbol{x}$ does not affect the distribution of $\boldsymbol{W}$. Secondly, for a given $\boldsymbol{W}$, $G(\boldsymbol{x}, \boldsymbol{W})$ can be easily evaluated for any $\boldsymbol{x}$. Thus, the observation of $G(\boldsymbol{x}, \boldsymbol{W})$ is *constructive*. As a result, we can apply the Sample Average Approximation (SAA) approach to $\mathcal{P}_1$ [15] to handle with uncertainty.

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

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

and the SAA problem $\mathcal{P}_2$ is defined as

$$\begin{aligned} \mathcal{P}_2: \quad & \min_{\boldsymbol{x}\in\mathcal{X}} \hat{g}_R(\boldsymbol{x}) \\ s.t. \quad & (9). \end{aligned}$$

By Monte Carlo Sampling, with support from *the Law of Large Numbers* [16], when R is large enough, the optimal value of $\hat{g}_R(\boldsymbol{x})$ can converge to the optimal value of $g(\boldsymbol{x})$ with probability one (w.p.1). As a result, we only need to care about how to solve $\mathcal{P}_2$ as optimal as possible.

4 Algorithm Design

In this section, we propose the SAA-RS algorithm. It includes a subroutine, named Genetic Algorithm-based Server Selection (GASS) algorithm. The details are presented below.

4.1 The SAA-RS Algorithm

The SAA-RS algorithm is presented in **Algorithm 1**. Firstly, we need to select the right sample size R and R'. 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, and often exponentially, in the sample size R [15]. 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' ($R' \gg R$). Secondlly, inspired by [15], we replicate generating and solving $\mathcal{P}_2$ with L i.i.d. replications. From Step. 4 to Step. 8, we call the subroutine GASS to obtain the approximate 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-RS returns the best solution among L replications and the algorithm terminates, otherwise we increase R and R' and drill again.

Algorithm 1 SAA-based Redundancy Scheduling (SAA-RS)

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

4.2 The GASS Subroutine

GASS is implemented based on the well-konwn Genetic Algorithm (GA). The reasons we choose GA can be summarized into several points as follows. Firstly, GA can approximate the optimal solution of the typical combinatorial optimization problem $\mathcal{P}_2$ within polynomial time. Secondly, the robustness can be guaranteed since it starts from a group of solutions. Thirdly, local optima can be avoided to a certain extent since the operation of crossover and mutation are executed randomly based on given parameters.

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 $\boldsymbol{x}_{p_1}$ and $\boldsymbol{x}_{p_2}$ are exchanged since the position $\boldsymbol{x}(b_j)$. 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 $\boldsymbol{x}_1, ..., \boldsymbol{x}_P \in \mathcal{X}$
3: **for** $t = 1$ to `it` **do**
4: $\forall p \in \{1, ..., P\}$, renew the optimization goal of $\mathcal{P}_2$, i.e. $\hat{g}_R(\boldsymbol{x}_p)$, according to (11)
5: **for** $p = 1$ to P **do**
6: **if** `rand()` $< \mathbb{P}_c$ **then**
7: Choose two chromosomes p_1 and p_2 according to the probability distribution:

$$\mathbb{P}(p \text{ is chosen}) = \frac{1/\hat{g}_R(\boldsymbol{x_p})}{\sum_{p'=1}^{P} 1/\hat{g}_R(\boldsymbol{x_{p'}})}$$

8: Randomly choose SBS $j \in \mathcal{M}$
9: Crossover the segement of $\boldsymbol{x}_{p_1}$ and $\boldsymbol{x}_{p_2}$ after the partitioning point $\boldsymbol{x}(b_{j-1})$:

$$[\boldsymbol{x}_{p_1}(b_j), ..., \boldsymbol{x}_{p_1}(b_M)] \leftrightarrow [\boldsymbol{x}_{p_2}(b_j), ..., \boldsymbol{x}_{p_2}(b_M)]$$

10: **end if**
11: **if** `rand()` $< \mathbb{P}_m$ **then**
12: Randomly choose SBS $j \in \mathcal{M}$ and re-generate the segement $\boldsymbol{x}_p(b_j)$
13: **end if**
14: **end for**
15: **end for**
16: **return** $\operatorname{argmin}_p \hat{g}(\boldsymbol{x_p})$ from P chromosomes

4.3 Guides for Implementation

SAA-RS can be periodically re-run to follow up end users' service demand pattern. During each period, we collect end users' microservice composition preferences (under authorization of privacy), and reconstruct pods based on the result from SAA-RS. This process can be carried out through rolling upgrade with Kubernetes.

The proposed algorithm can replace the default scheduler of Kubernetes, i.e., `kube-scheduler`. It 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 [14]. 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. In conclusion, the default scheduler places microservices only based on resources occupancy of nodes. Our algorithm takes both the service request pattern of mobile users and the heterogeneity of the distributed nodes into consideration. However, in order to run SAA-RS in a non-simulated way, we need a real-world edge computing scenario. We left this task to our future work. In this paper, we only conduct our experiments in a simulated way.

5 Theoretical Analysis

In this section, we demonstrate the convergence of SAA method, which includes not only the convergence of the optimization goal and the corresponding solution of $\mathcal{P}_2$, but also the convergence rate. More detailed analysis and proofs can be consulted in professional articles and books, such as [15] [17].

Recall that the domain $\mathcal{X}$ of problem $\mathcal{P}_1$ and $\mathcal{P}_2$ is finite, whose size is $\prod_{j\in\mathcal{M}} b_j\big(\sum_{q\in\mathcal{Q}} C_q + 1 - \frac{1}{2}(b_j - 1)\big)$. Thus, $\mathcal{P}_1$ and $\mathcal{P}_2$ 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 $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. To analysis the optimality, we also define the set of ϵ-optimal solutions.

Definition 5.1. **ϵ-optimal Solutions** For $\epsilon \geq 0$, if $\boldsymbol{x} \in \mathcal{X}$ and $g(\boldsymbol{x}) \leq v^* + \epsilon$, then we say that $\boldsymbol{x}$ is an ϵ-optimal solution of $\mathcal{P}_1$. Similarly, if $\boldsymbol{x} \in \mathcal{X}$ and $g(\boldsymbol{x}) \leq \hat{v}_R + \epsilon$, $\boldsymbol{x}$ is an ϵ-optimal solution of $\mathcal{P}_2$.

We use $\mathcal{X}^\epsilon$ and $\hat{\mathcal{X}}_R^\epsilon$ to denote the set of ϵ-optimal solutions of $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. Then we have the following proposition.

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

Proof 5.1. It can be directly obtained from *Proposition 2.1* of [15], not tired in words here.

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

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 satisify

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

where

$$\sigma_{max}^2 \triangleq \max_{\boldsymbol{x}\in\mathcal{X}\backslash\mathcal{X}^\epsilon} \mathrm{Var}\big[G(u(\boldsymbol{x}), \boldsymbol{W}) - G(\boldsymbol{x}, \boldsymbol{W})\big],$$

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

Proof 5.2. It can be directly obtained by combing *Proposition 2.2* of [15] with the size of $\mathcal{X}$. For more details, please consult [15] directly.

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

6 Experimental Evaluation

In this section, we evaluate both the efficiency of redundancy scheduling and the optimality of SAA-RS through simulations. Section 6.1 introduces several benchmarks. Section 6.2 demonstrates the basic settings of global parameters. The experimental results are analyzed in Section 6.3.

6.1 Benchmark Policies

Besides evolutionary algorithm, we also consider two representative benchmarks, i.e., random algorithm and greedy policy to evaluate our policy under two scenarios. In the first scenario, redundancy is not allowed. Each candidate can only be dispatched to only one SBS. The first scenario is used to evaluate the correctness and rationality of redundancy scheduling. In the second scenario, redundancy is allowed. In this case, those benchmarks are used to replace GASS to generate the best-so-far solution of each sampling. This scenario is used to evaluate the optimality and efficiency of GASS. In both scenarios, those benchmarks will be run R times and the average value is returned. Details are summarized as follows.

(1) **Random Scheduling in Scenario #1 (RS1)**: $\forall q \in \mathcal{Q}, \forall c_q \in \mathcal{C}_q$, dispatch c_q to a randomly chosen SBS j and decrement b_j. The procedure terminates if no available SBS.

(2) **Random Scheduling in Scenario #2 (RS2)**: $\forall q \in \mathcal{Q}, \forall c_q \in \mathcal{C}_q$, 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)**: Encode vector $[\mathbb{1}\{s^{c_1}(t_1)\}, ..., \mathbb{1}\{s^{cc_Q}(t_Q)\}]^\top$ as a chromosome, where $\mathbb{1}\{s^{c_q}(t_q)\}$ means that $s^{c_q}(t_q)$ is chosen to deploy. Randomly generate the initial population of chromosomes across the searching space. Create the offspring population by selection, recombination, and mutation. Only those chromosomes with better fitness can propagate.

(4) **Greedy Policy in Scenario #2 (GP2)**: The greedy policy is embodied in three perspectives. Firstly, GP2 maximizes the utility of SBSs' resources, i.e., each SBS j will be deployed b_j candidates, rather than an integer between 0 and b_j. Secondly, each mobile device always chooses the nearest available SBS to execute their selected candidates. Thirdly, each SBS will only deploy the most frequently selected candidate in each iteration. Details of this benchmark is demonstrated in **Algorithm 3**.

Algorithm 3 Greedy Policy in Scenario #2 (GP2)

availSBSs $\leftarrow \mathcal{M}$
$\forall q \in \mathcal{Q}$, needSchedule$[q] \leftarrow \mathcal{N}$
$\forall q \in \mathcal{Q}, \forall c_q \in \mathcal{C}_q, \mathcal{D}(s^{c_q}(t_q)) \leftarrow \varnothing$
while availSBSs $\neq \varnothing$ **do**
  for $q = 1$ to Q **do**
    $\forall j \in \mathcal{M}$, containers$[j] \leftarrow \varnothing$
    for $i \in$ needSchedule$[q-1]$ **do**
      if $q == 1$ **then**
        Get the nearest $j^* \in$ availSBSs to $j^\star(s^{c_1^i}(t_1))$ (could be the same one)
      else
        Calculate $j_p(s^{c_{q-1}^i}(t_{q-1}))$ by (4) or (5)
        Get the nearest SBS j^* to $j_p(s^{c_{q-1}^i}(t_{q-1}))$ from availSBSs (could be the same one)
      end if
      Add $s^{c_q^i}(t_q)$ to containers$[j^*]$
    end for
    for $j \in$ availSBSs **do**
      Find the most frequent candidate c of microservice t_q in containers$[j]$, add it to $\mathcal{D}(s^{c_q}(t_q))$
      $b_j \leftarrow b_j - 1$
      if $b_j == 0$ **then**
        remove j from availSBSs
      end if
      For those mobile devices who choose c to execute t_q, remove them from needSchedule$[q-1]$
    end for
  end for
end while
return $\mathcal{D}(s^{c_q}(t_q)), \forall q \in \mathcal{Q}, \forall c_q \in \mathcal{C}_q$

6.2 Experimental Settings

The experiment is conducted based on the geolocation information about 125 edge sites and 817 end users within the Melbourne CBD area contained in the EUA dataset [18]. As demonstrated in Fig. 2, we choose one edge site from the dataset as the MBS to provide ubiquitous signal coverage. Unless otherwise specified, the coverage radius of each SBS is sampled from $[100, 400]$ meters uniformly. In our simulations, unless otherwise specified, N is setted as 200, M is setted as 20, and the maximum hops between any two SBSs can not larger than 4. In additon, $\forall j \in \mathcal{M}$, b_j is chosen from $\{b_j^{min}, ..., b_j^{max}\}$ uniformly, where $b_j^{min} = 1, b_j^{max} = 3$ in default. For the considered application, we set Q as

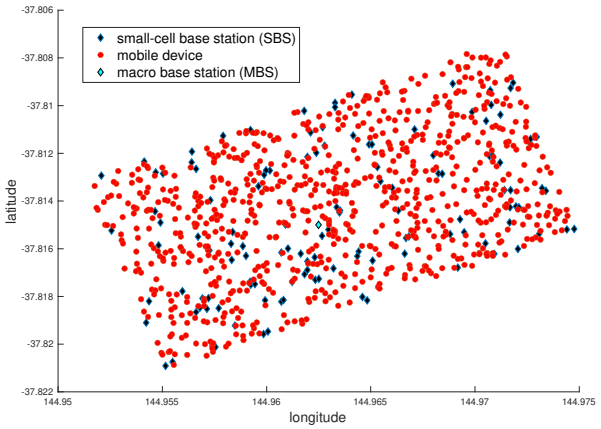


Fig. 2. Mobile devices, SBSs, and MBS in our simulation.

4, and C_q as $3, 4, 2, 4$ when $q = 1, 2, 3, 4$, respectively. $\forall c_1 \in \mathcal{C}_1, \mathbb{P}(\boldsymbol{E}(s^{c_1}(t_1))$ is setted as

$$\begin{bmatrix} 0.2 & 0.3 & 0.5 \end{bmatrix},$$

$\forall c_1 \in \mathcal{C}_1, c_2 \in \mathcal{C}_2, \mathbb{P}(\boldsymbol{E}(s^{c_2}(t_2)) | \boldsymbol{E}(s^{c_1}(t_1))$ is setted as

$$\begin{bmatrix} 0.3 & 0.4 & 0.1 & 0.2 \\ 0.4 & 0.2 & 0.3 & 0.1 \\ 1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix},$$

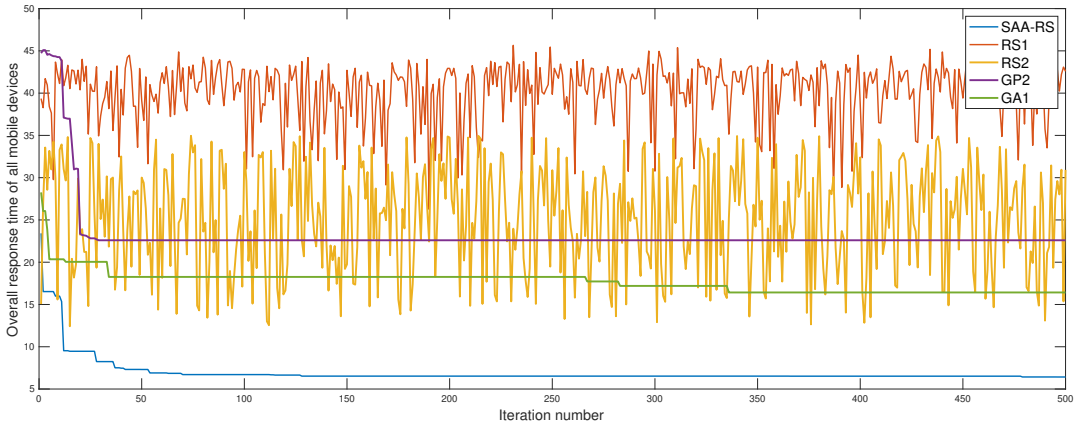


Fig. 3. Overall response time of all mobile devices vs. iteration, $N = 200$ and $M = 20$.

$\forall c_2 \in \mathcal{C}, c_3 \in \mathcal{C}_3, \mathbb{P}(\boldsymbol{E}(s^{c_3}(t_3)|\boldsymbol{E}(s^{c_2}(t_2))$ is setted as

$$\begin{bmatrix} 0.5 & 0.5 \\ 0.3 & 0.7 \\ 0.6 & 0.4 \\ 0.1 & 0.9 \end{bmatrix},$$

and $\forall c_3 \in \mathcal{C}_3, c_4 \in \mathcal{C}_4, \mathbb{P}(\boldsymbol{E}(s^{c_4}(t_4)|\boldsymbol{E}(s^{c_3}(t_3))$ is setted as

$$\begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 \\ 0.6 & 0.2 & 0.1 & 0.1 \end{bmatrix}.$$

In our settings, the third candidate of microservice #1 *is correlated with* the first candidate of microservice #2. The basic settings of other system parameters are presented in Table 2 unless otherwise specified. 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.

TABLE 2
System parameters.

Parameter	Value	Parameter	Value
α	0.08 ms	β	5 ms
τ_b	0.1 s	L	5
R	500	R'	100000
τ_{exe}^{min}	1 ms	τ_{exe}^{max}	2 ms
population	10	generation	200
$\mathbb{P}_m$	10%	$\mathbb{P}_c$	80%

6.3 Experiment Results

6.3.1 Optimality and Stability

As demonstrated in Fig. 3, SAA-RS outperforms other four benchmarks significantly in the overall response time of all mobile devices, i.e., $\sum_{i\in\mathcal{N}} \tau(\boldsymbol{E}(s^{c_q^i}(t_q))_{q=1}^{Q})$. Within 500 iterations, SAA-RS outperforms GA1, GP2, RS2, and RS1 by 60.89%, 71.51%, 79.23%, and 84.91%, respectively. The result verifies both the rationality of redundancy scheduling and the optimality of our algorithm. Firstly, it shows that SAA-RS can converge to an approximate optimal solution, i.e. 6.5259 ms at a rapid rate (at the 128th iteraiton). The achieved solution significantly superior to benchmarks in scenario #2. Secondly, redundancy scheduling is effective, which is manifested by the results achieved by RS1 and RS2. Under the same circumstances, RS2 outperforms RS1 by 37.55% in average. Actually, the superiority of redundancy scheduling lies in that it takes full advantage of the distributed edge servers' limited resources. In this case, the processing of end users' service requests can surely be balanced.

In addition to the above phenomena, it is interesting to see that GP2 is inferior to GA1 by 27.32%. Actually, 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. However, GP2 also has the common problem of greedy policy, i.e. nearsightedness. GP2 cannot modify existing deployment even a better soluiton is appeared. As shown in Fig. 3, GP2 at most iterates $\sum_{j\in\mathcal{M}} b_j$ times and no improvement can be made (22.5955 ms at the 29th iteraiton). In contrast, GA1 can continuously improve the best-so-far solution through crossover and mutation.

Fig. 4 and Fig. 6 demonstrate the comparison between SAA-RS with different number of mobile devices N and different number of SBSs M, respectively. As demonstrated in Fig. 4, for all implemented algorithms, the overall response time increase as N increases. When N increases from 100 to 800, the solution achieved by SAA-RS is always the best. It is interesting that besides RS2, the gaps between those benchmarks and SAA-RS increase as N increases. Specifically, for GA1, the gap increases from 11.75% to 21.78%; for RS2, the gap decreases from 75.17% to 51.84%; for GP2, the gap increases from 61.19% to 77.75%; for RS1, the gap increases from 80.36% to 82.89%. It indicates that SAA-RS is robust to the computation complexity of the fitness function. Besides, we can see that when $N > 200$, GP2 is surpassed by RS2. It indicates that the weakness of greedy policy is magnified by the increase of N.

Fig. 6 demonstrates the impact of M on the optimality. Similarly, the superiority of SAA-RS is obvious, as it is always the best of five algorithms whatever M takes. The gaps

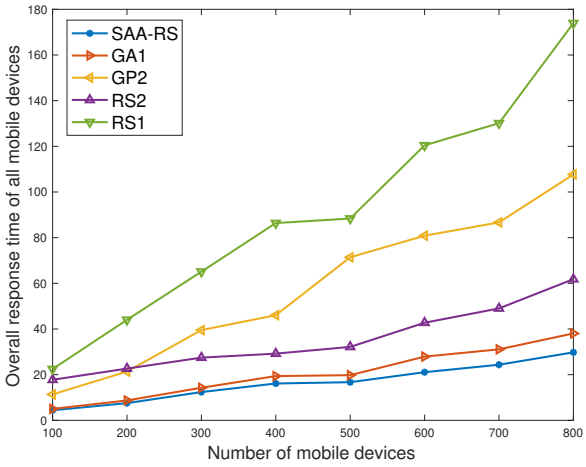


Fig. 4. Overall response time of all mobile devices vs. N, $M = 20$.

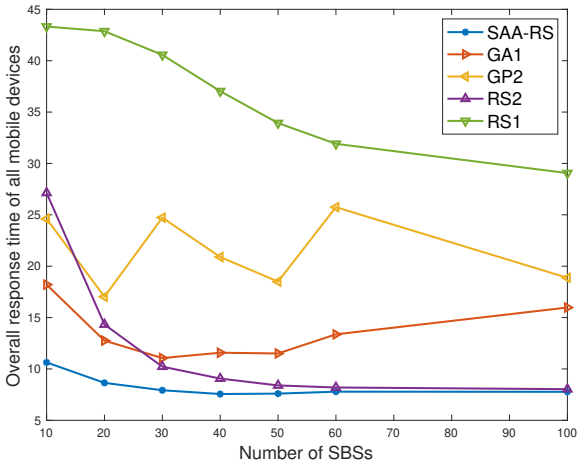


Fig. 6. Overall response time of all mobile devices vs. M, $N = 200$.

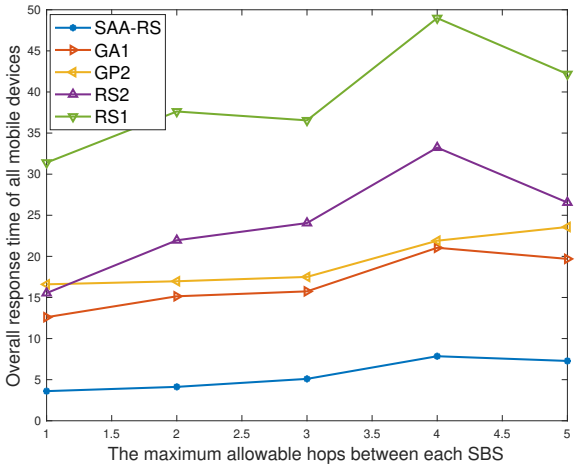


Fig. 5. Overall response time of all mobile devices vs. sparsity of the graph, $M = 10$.

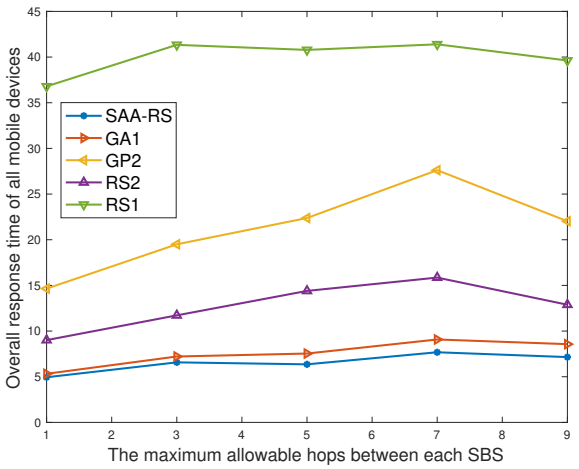


Fig. 7. Overall response time of all mobile devices vs. sparsity of the graph, $M = 20$.

between SAA-RS and RS2, GA1, GP2, and RS1 ranges between $[7.79\%, 60.81\%]$, $[28.32\%, 51.35\%]$, $[49.22\%, 69.75\%]$, and $[75.45\%, 75.64\%]$, respectively. Over all, as M increases, the response time decreases. This is because more SBSs can provide more resources, which greatly helps to realize *near-request processing*. However, several details need more nuanced analysis. Firstly, the response time of both SAA-RS and GA1 has a slight rising trend (2.43% and 27.98%, respectively). This is because when M increases, the dimension of feasible solution increases, which greatly expands the solution space. Under this circumstances, 200 iterations might not be enough to achieve an optimal enough solution. Besides, the sparsity of the connected graph decreases as M increases, which brings in more locally optimal solutions and reduces the optimization space. Secondly, RS2 is able to achieve a better solution than GA1 when M is large (47.23% in superiority when $M = 100$), which greatly proves the worth of redundancy scheduling.

Fig. 5 and Fig. 7 demonstrates the impact of the sparsity of the graph. In order to simplify the experiment design, let us use *the maximum allowable hops* between any two SBSs to represent the sparsity of the graph. First of all, SAA-RS is stably superior to others, under any circumstances. The gaps between these benchmarks and SAA-RS is larger when the number of SBSs is smaller. This is because appropriate increase of SBSs can reduce the processing load of servers. Specifically, when $M = 10$, the gaps between SAA-RS and RS2, GA1, GP2, and RS1 ranges between $[71.3\%, 76.79\%]$, $[63.04\%, 71.39\%]$, $[69.13\%, 78.28\%]$, and $[82.74\%, 88.52\%]$, respectively; when $M = 20$, the gaps between SAA-RS and RS2, GA1, GP2, and RS1 ranges between $[45.11\%, 51.57\%]$, $[7.12\%, 15.42\%]$, $[66.24\%, 72.19\%]$, and $[81.45\%, 86.56\%]$, respectively. Another interesting phenomenon happens in both Fig. 5 and Fig. 7 is that the overall response time increases at the early stage and then decrease when the graph is more and more sparse. It happens to almost all five algorithms. Actually, this is a trade-off between the size of optimizable solution space and the time consumption on routing. When the graph is dense, the routing consumption is small in general. In this case, the approximate optimality

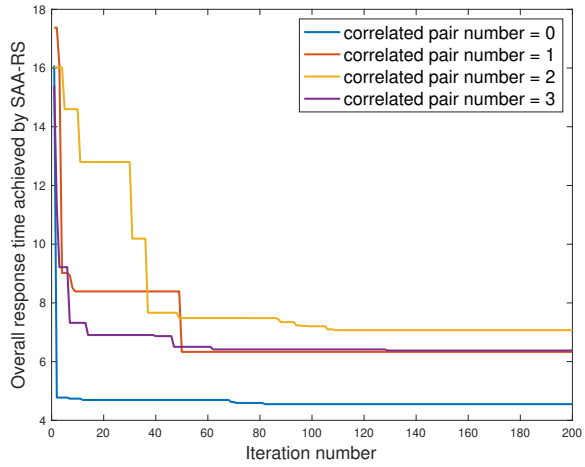


Fig. 8. Overall response time of all mobile devices vs. iteration.

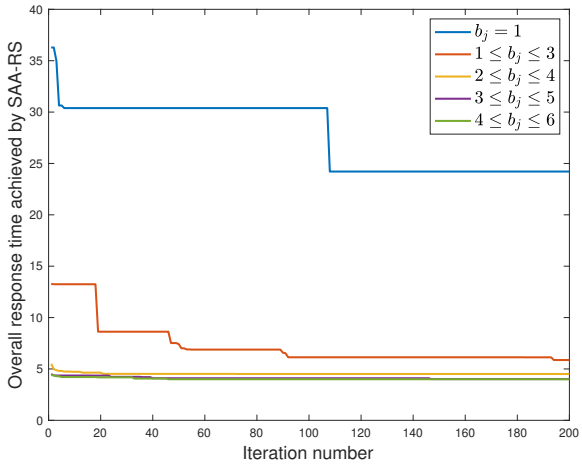


Fig. 10. Overall response time of all mobile devices vs. iteration..

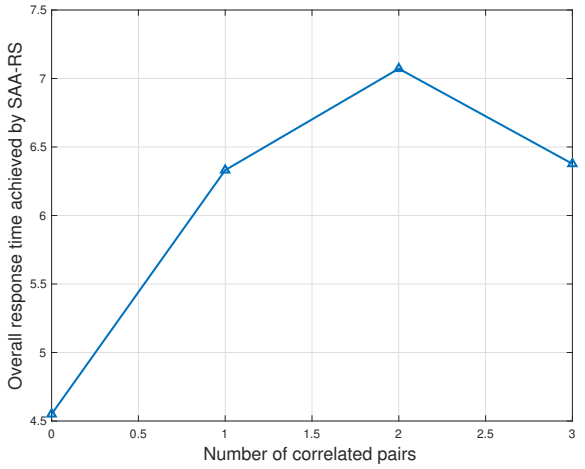


Fig. 9. Overall response time of all mobile devices vs. sparsity of the graph vs. number of correlated pairs.

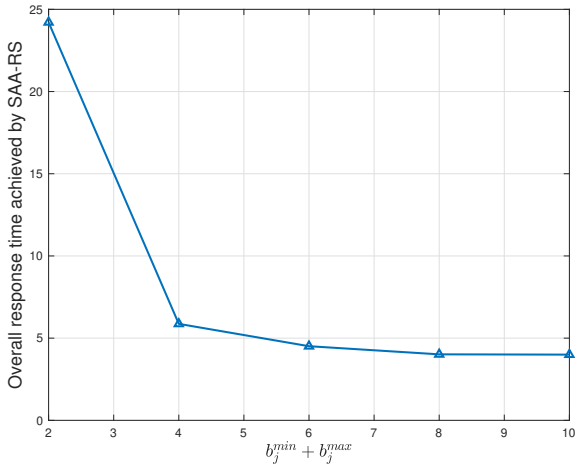


Fig. 11. Overall response time of all mobile devices vs. b_j.

is easy to obatined. When the graph is sparse enough, although the routing time is large in general, the optimization solution space is relatively small becasue the variability of fitness decreases. Under this circumstance, the overall response time is still small.

6.3.2 *Impacts of Important Parameters*

In this Section, we analyze the impacts of important parameters in both model formulation and algorithms design. For model formulation, we emphasize *the correlation of microservices* and *the maximum deployable microservice instances*. Let us use the number of correlated pairs to analyze its impacts. As we have setted before, $Q = 4$. Thus, for the most, the number of correlated pairs is 4. Fig. 8 and Fig. 9 demonstrate its impacts on the performance of SAA-RS. When there is no correlated microservices, the overall response time is the least. But no distinct difference can be found. However, the situation is completely different for the maximum deployable instances, i.e. b_j. As we can see from Fig. 10 and Fig. 11, the performance of SAA-RS is negatively correlated with b_j. The result is obvious becasue b_j decides the upper limit of resources, which is the key influence factor of overall response time.

Fig. 12, FIg. 13, and Fig. 14 analyze the impacts of population size, the probability of mutation, i.e. $\mathbb{P}_m$, and the probability of crossover, i.e. $\mathbb{P}_c$, respectively. For SAA problem $\mathcal{P}_2$, their impacts on the performance of SAA-RS is minor. As a result, no more detialed discussion is launched.

7 Related Works

Service computing based on traditional cloud datacenters has been extensively studied in the last several years, especially service selection for composition [19] [20] [21], service provision [22], discovery [23], recommendation [24], 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.

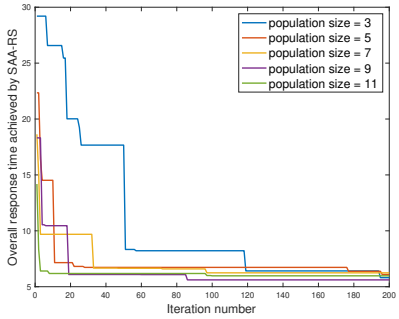


Fig. 12. Impacts of population size.

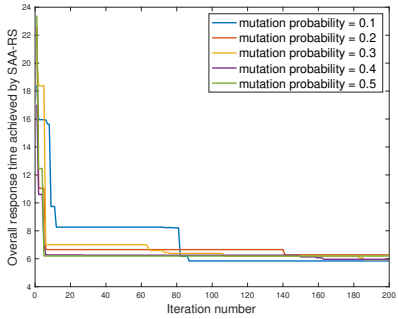


Fig. 13. Impacts of $\mathbb{P}_m$.

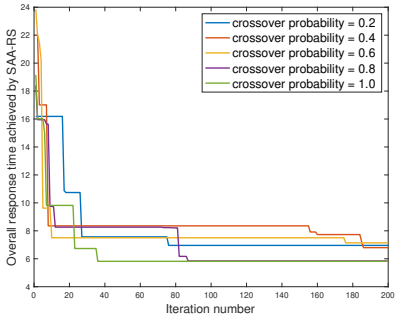


Fig. 14. Impacts of $\mathbb{P}_c$.

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 [5]. 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 [8]. 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 [6] [7] [9]. 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 redundancy scheduling 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 [25] [26] [27]. 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 [13]. 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 bacause 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 [28]. To improve the practicability, we give the deployment details with container technologies and Kubernetes in the typical pre-5G HetNet. Besides, we analyze the response time of each mobile devices in a more rigorous manner, and improve it by always finding the nearest available edge sites. 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 embedd the idea of redundancy into the problem and design a new algorithm with a faster convergence rate.

8 Conclusion

In this paper, we study a redundancy scheduling policy for the deployment of microservice-based applications at the distributed edge. We first demonstrate a pre-5G HetNet, and then explore the possibilities of the deployment of composite microservices with containers and Kubernetes. After that, we model the redundancy scheduling as a stochastic optimization problem. For the application with composite and correlated microservices, we design a SAA-based algorithm SAA-RS and its subroutine GASS to dispatch microservice instances into edge sites. By creating multiple access to services, our policy boosts a faster response for mobile devices significantly. SAA-RS 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 efficiency of redundancy scheduling and the optimality of SAA-RS. In addition, we give guidance on the implementation of SAA-RS with Kubernetes. In our future work, we will hammer at the implementation of the redundant deployment in a real-world edge computing scenario.

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] "Docker: Modernize your applications, accelerate innovation." [Online]. Available: https://www.docker.com/

[4] "Kubernetes: Production-grade container orchestration." [Online]. Available: https://kubernetes.io/

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] F. Ait Salaht, 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] "Kubernetes scheduler." [Online]. Available: https://kubernetes.io/docs/concepts/scheduling/kube-scheduler/

[15] 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.

[16] C. Robert and G. Casella, *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

[17] B. K. Pagnoncelli, S. Ahmed, and A. Shapiro, "Sample average approximation method for chance constrained programming: theory and applications," *Journal of optimization theory and applications*, vol. 142, no. 2, pp. 399–416, 2009.

[18] 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.

[19] 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.

[20] S. Wang, A. Zhou, R. Bao, W. Chou, and S.-S. Yau, "Towards green service composition approach in the cloud," *IEEE Transactions on Services Computing*, Aug 2018.

[21] S. K. Gavvala, C. Jatoth, G. Gangadharan, and R. Buyya, "Qos-aware cloud service composition using eagle strategy," *Future Generation Computer Systems*, vol. 90, pp. 273 – 290, 2019.

[22] 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.

[23] 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.

[24] X. Chen, Z. Zheng, Q. Yu, and M. R. Lyu, "Web service recommendation via exploiting location and qos information," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1913–1924, July 2014.

[25] H. Deng, T. Zhao, and I. Hou, "Online routing and scheduling with capacity redundancy for timely delivery guarantees in multihop networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 1258–1271, June 2019.

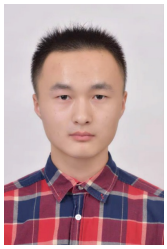
[26] A. abdi, A. Girault, and H. R. Zarandi, "Erpot: A quad-criteria scheduling heuristic to optimize execution time, reliability, power consumption and temperature in multicores," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2193–2210, Oct 2019.

[27] 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.

[28] 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 an 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).