It's Hard to Share: Joint Service Placement and Request Scheduling in Edge Clouds with Sharable and Non-sharable Resources

Ting He*, Hana Khamfroush[†], Shiqiang Wang[‡], Tom La Porta*, and Sebastian Stein[§]
*Pennsylvania State University, University Park, PA, USA. Email: {t.he,tlp}@cse.psu.edu
[†]University of Kentucky, Lexington, KY, USA. Email: hanakhamfroush@gmail.com
[‡]IBM T. J. Watson Research Center, Yorktown, NY, USA. Email: wangshiq@us.ibm.com
[§]University of Southampton, Southampton, UK. Email: ss2@ecs.soton.ac.uk

***Abstract*—Mobile edge computing is an emerging technology to offer resource-intensive yet delay-sensitive applications from the edge of mobile networks, where a major challenge is to allocate limited edge resources to competing demands. While prior works often make a simplifying assumption that resources assigned to different users are non-sharable, this assumption does not hold for storage resources, where users interested in services (e.g., data analytics) based on the same set of data/code can share storage resource. Meanwhile, serving each user request also consumes non-sharable resources (e.g., CPU cycles, bandwidth). We study the optimal provisioning of edge services with non-trivial demands of both sharable (storage) and non-sharable (communication, computation) resources via joint service placement and request scheduling. In the homogeneous case, we show that while the problem is polynomial-time solvable without storage constraints, it is NP-hard even if each edge cloud has unlimited communication or computation resources. We further show that the hardness is caused by the service placement subproblem, while the request scheduling subproblem is polynomial-time solvable via maximum-flow algorithms. In the general case, both subproblems are NP-hard. We develop a constant-factor approximation algorithm for the homogeneous case and efficient heuristics for the general case. Our trace-driven simulations show that the proposed algorithms, especially the approximation algorithm, can achieve near-optimal performance, serving 2–3 times more requests than a baseline solution that optimizes service placement and request scheduling separately.**

***Index Terms*—mobile edge computing; service placement; request scheduling; complexity analysis; approximation algorithm.**

I. Introduction

Mobile edge computing [1], [2] has been one of the fastest-rising technology trends for offering resource-intensive yet delay-sensitive applications from the edge of mobile networks. Compared to the approaches of running such applications on mobile devices or servers located deep in the Internet, mobile edge computing takes the approach of running these applications on small cloud-computing platforms deployed at the network edge (e.g., access points or base stations), referred to as *edge clouds* [3] (a.k.a. *cloudlets* [4], *fog* [5], *follow me cloud* [6], and *micro clouds* [7]). Mobile edge computing allows users to exploit the power of cloud computing without incurring the high latency in communicating to remote clouds.

A major limitation in mobile edge computing is that compared with traditional data centers, each edge cloud (consisting of a single server or a small cluster of co-located servers) is much more limited in resources. With the help of virtualization techniques such as virtual machines (VM) [8], [9] and containers [10], [11], the research community, industry players, and standardization bodies have formed several initiatives [12], [13], [14] to create a standardized open edge computing environment, such that edge clouds within the same geographical region will form a shared resource pool. The main question is how to optimally allocate resources from this pool to competing demands.

While having received significant attention in recent years, existing solutions [15], [16], [3], [17], [18] mostly assume additive resource consumption, i.e., the total resource consumption on an edge cloud is the sum of all the demands scheduled to it. While this assumption holds for resources like CPU cycles and link bandwidth, where each user request is served by a dedicated slice of resource, it fails to fully characterize the demands of data analytics services. In data analytics services such as augmented reality and video analytics [19], serving each request requires both a non-trivial amount of computation and data on the server, and a non-trivial amount of data from the user. For example, in augmented reality, the server needs to store the object database and the (pre-trained) visual recognition models, and the user needs to upload the current camera view, on which the server runs classification/object recognition and returns the augmented information in real time.

When providing such services to a large number of users, it may not always be possible to serve all the users from the network edge, or serve every user from its closest edge cloud. As was shown in [20], [21], [22], users can benefit from resource pooling at the scale of metropolitan area networks (MAN), which allows users to access services on edge clouds outside their one-hop communication range. In terms of resource consumption, users querying against the same server dataset (e.g., object database related to the same region) can be served by the same service replica, while serving each

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

user request (e.g., augmenting a specific view) consumes a dedicated share of CPU and bandwidth (for uploading the camera view). The fundamental problem in provisioning such services from the network edge is how to optimally place services in edge clouds and schedule user requests to the placed services, while *considering both sharable and non-sharable resources*[1]. While augmented reality and video analytics have been identified by ETSI as examples of high-value applications for mobile edge computing [19], the same problem applies to any applications where users interested in the same service can share the same copy of data/code, while individual service requests compete for CPU and bandwidth.

Treating each type of data analytics as a "service", we address the above problem by formulating an integer linear program (ILP) aiming at serving the maximum user requests by a given pool of edge clouds, each with limited communication, computation, and storage capacities. We jointly study two related subproblems: (1) *service placement*, which determines where to place each service (including data and code), allowing multiple replicas per service, within the storage capacities of edge clouds, and (2) *request scheduling*, which determines whether/where to schedule each request subject to wireless communication capacities between users and edge clouds, computation capacities of edge clouds, and other feasibility constraints (e.g., maximum tolerable latency), as well as a constraint that the edge cloud scheduled to process a request must have a replica of the requested service.

A. Related Work

Research on mobile edge computing has evolved to allow users to access services on edge servers that are not within their one-hop communication range [20], [21], [22]. These works have proposed algorithms to place edge servers and assign users to the servers, so that all the users can be covered within a reasonable delay.

Allowing a user to access multiple edge servers opens up the problem of workload scheduling. There is a rich literature on edge workload scheduling, with various objectives (e.g., minimizing the cost [18] or the makespan [23]), workload models (e.g., fluid model [18], tasks [23], multi-component applications [24]), and edge cloud architectures (e.g., flat versus hierarchical [25]). These works typically assume that each workload requires its own resource for execution, i.e., the resources are non-sharable. While this assumption usually holds for computation and communication resources (assuming unicasts), it can be too restrictive for storage resources. Note that although [23] allows each service replica to serve multiple jobs, it does not optimize the service placement.

Another line of related work is content placement in cache networks under cache capacity constraints, based on predicted content popularities [26], [27] or request history [28]. In particular, the content placement problem has been studied in a cooperative cache cluster where a cache can serve requests from other caches [29]. The problem of joint content placement and request routing has also been studied [30]. However, the content placement problem only focuses on the sharable resource of cache space, ignoring the other types of resources (e.g., CPU, bandwidth) consumed additively in serving individual user requests. Although [28] was motivated by "hosting services", the solution was actually about caching.

Multiple types of resources (e.g., storage, computation, communication) have been considered in the content/service distribution problem [31], [32]. In [31], a mixed integer linear program (MILP) was formulated for placing service functions and activating storage, computation, and communication resources in a distributed cloud network. In [32], a similar MILP was formulated for placing content. However, no formal complexity analysis or algorithm with performance guarantee was provided. A dynamic service placement and workload scheduling framework was proposed in [33] to jointly consider storage and computation resources, but there is no hard constraint on computation resources and no consideration of bandwidth limitation. Recently, [34] proposed an algorithm with performance guarantee for placing virtual network functions (VNFs) in distributed cloud networks and routing service flows among the placed VNFs under chaining constraints. However, all the resources (CPU, memory, bandwidth) are consumed additively by service flows (i.e., non-sharable). A work ostensibly similar to ours is [35], which studied joint resource placement and assignment in distributed networks with multiple resource types. However, the work assumed that each placed resource can only serve one request, i.e., non-sharable, which is critically different from our problem that considers both sharable and non-sharable resources, leading to very different conclusions: the problem in [35] is polynomial-time solvable, but our problem is NP-hard even in several special cases (Section III).

To our knowledge, we are the first to rigorously study joint service placement and request scheduling in mobile edge computing, while simultaneously considering hard constraints on both communication/computation resources that are consumed additively and storage resources that are sharable among requests of the same type.

B. Summary of Contributions

We consider the problem of *joint service placement and request scheduling* in edge clouds. Our contributions are:

1) We formulate the problem as an integer linear program (ILP) aiming at serving the maximum number of user requests using limited communication, computation, and storage resources per edge cloud, considering that the storage resource may be shared by requests of the same service.

2) We analyze the complexity of the problem in both the general case and important special cases. We show that not only the general case is NP-hard, but the special case of homogeneous edge clouds and homogeneous services is also NP-hard, and the hardness remains even in more special cases when each edge cloud has unlimited communication or computation resources. We further show that in the homogeneous case, the hardness is caused by the service placement subproblem, while in the general case, both the service placement subproblem and the request scheduling subproblem are NP-hard.

3) We propose polynomial-time solutions. In the homogeneous case, we show that the optimal resource scheduling (un-

[1]Here, *sharable* means a unit of resource can serve more than one unit of demand, and *non-sharable* means a unit of resource can only serve a unit of demand (even if the physical resource, e.g., CPU/link, can be shared temporally/spectrally).

TABLE I
NOTATIONS

Symbol	Meaning
N	set of edge clouds
U	set of users (user $\leftrightarrow$ user request)
L	set of all possible services
$n_u \in N$	edge cloud covering user u
$l_u \in L$	service requested by user u
$a_{un} \in \{0, 1\}$	whether edge cloud n is allowed to serve user u
U_n	set of users covered by edge cloud n
K, W, R	communication, computation, and storage capacities per edge cloud (homogeneous case)
$x_{ln} \in \{0, 1\}$	indicator of placing service l at edge cloud n
$y_{un} \in \{0, 1\}$	indicator of scheduling user u to edge cloud n

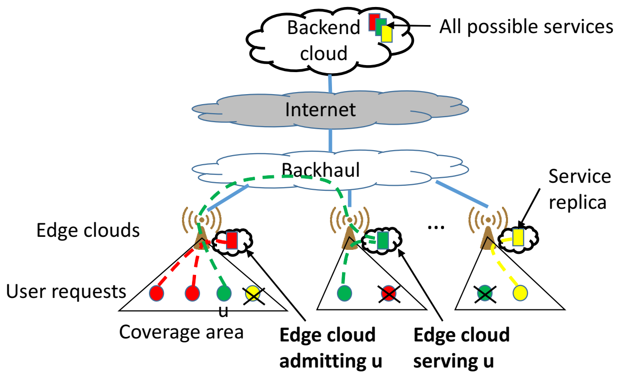


Fig. 1. System model, where different colors represent different types of services ($R = 1$, $W = 2$, $K = 3$).

der a given service placement) can be computed in polynomial time by converting it to a maximum flow problem. We further show that under certain conditions, combining this solution with greedy service placement gives an overall solution with $1/2$-approximation to the optimum. In the general case, we develop a heuristic that combines greedy service placement with greedy request scheduling, and a heuristic based on linear program (LP) relaxation and rounding.

4) We evaluate the proposed solutions via trace-driven simulations. Our results show that the proposed solutions can serve 2–3 times more requests than a baseline solution that separately optimizes service placement and request scheduling. Moreover, the approximation algorithm achieves near-optimal performance, even under prediction errors in user demands.

Roadmap. Section II formulates the problem in the homogeneous case. Section III analyzes the complexity of the problem, and Section IV presents the proposed solutions. Section V extends the formulation and the solutions to the heterogeneous case. Section VI evaluates our solutions against benchmarks. Finally, Section VII concludes the paper.

II. PROBLEM FORMULATION

We start by formally defining our model of the mobile edge computing system and formulating our resource provisioning problem. Table I summarizes the main notations, where the last two are decision variables and the rest are input parameters.

A. System Model

As illustrated in Fig. 1, we consider a mobile edge computing system consisting of a set of edge clouds N, a set of services L (each encapsulating the data and the code for the service), and a set of user-generated service requests U. Each edge cloud is assumed to be associated with a wireless access point (e.g., base station, WiFi access point) covering a local area referred to as a cell. We assume that the cells collectively cover all the users (as uncovered users have to be ignored) and each user is associated with only one cell. For simplicity, we will refer to a user request as a "user". The same user generating multiple requests will be modeled as multiple collocated users (possibly requesting different services).

Each edge cloud has certain communication, computation, and storage resources, and the edge clouds in N form a resource pool that serves the users *collaboratively*. We assume that the edge clouds are connected by backhaul links that can be used to send requests/responses between edge clouds, which allows a user to be served by a non-local edge cloud. Specifically, let $n_u \in N$ denote the edge cloud *covering* (i.e., directly associated with) user u, and $l_u \in L$ denote the service requested by user u. We allow the user to be served from any edge cloud within a *candidate set* $N_u \subseteq N$, as long as the resource constraints specified below are satisfied. Parameter N_u is used to capture predetermined requirements on candidate servers, such as quality of service requirements (e.g., edge clouds reachable within a certain latency), hardware requirements (e.g., edge clouds with GPU), and security requirements. For ease of presentation, we encode N_u by indicators $\{a_{un}\}_{n \in N}$, where $a_{un} = 1$ if and only if $n \in N_u$. Let $U_n \triangleq \{u \in U : n_u = n\}$ be the set of users covered by edge cloud n.

We consider three types of resource constraints: (1) the *communication capacity* of the wireless access point associated with each edge cloud limits the number of users the edge cloud can admit, i.e., receive requests and return responses for[2], (2) the *computation capacity* of each edge cloud limits the number of users the edge cloud can serve, i.e., process requests for, and (3) the *storage capacity* of each edge cloud limits the number of service replicas the edge cloud can store (and hence the number of different services it can provide). As a starting point, we consider the homogeneous case with identical capacities for all the edge clouds and identical demands for all the users, which is already challenging as shown in Section III. In this case, the communication capacity is simply measured by the maximum number of users admitted by an edge cloud, denoted by K; the computation capacity is the maximum number of users served by an edge cloud, denoted by W; the storage capacity is the maximum number of different services an edge cloud can provide, denoted by R. Note that only the edge cloud n_u directly covering a user u can admit it into the mobile edge computing system, but any edge cloud in the candidate set N_u can serve its request. We will address the heterogeneous case in Section V.

[2]We focus on communication capacity constraints imposed by the **last-hop wireless communications** between the users and their access points instead of the inter-edge cloud communications, because the last-hop capacity is typically much smaller than the capacity in the backhaul.

B. Optimization Formulation

Our goal is to optimally provision the services such that we can serve the maximum number of users (i.e., user requests) from the network edge within the resource constraints. As illustrated in Fig. 1, this includes determining which services to store at each edge cloud, i.e., *service placement*, and how to schedule users to edge clouds hosting their requested services, i.e., *request scheduling*. The latter also implies admission control, as users not scheduled to any edge cloud will be dropped. Here "dropping" a user just means not serving it from the edge; a dropped user can be served from the backend cloud, typically at a higher (performance/operational) cost.

To model the above decisions, we introduce two sets of decision variables: $x_{ln} \in \{0,\ 1\}$ ($l \in L,\ n \in N$), which indicates whether service l is placed at edge cloud n, and $y_{un} \in \{0,\ 1\}$ ($u \in U,\ n \in N$), which indicates whether user u is scheduled onto edge cloud n. Here $\mathbf{x} \triangleq (x_{ln})_{l\in L, n\in N}$ denotes the service placement, and $\mathbf{y} \triangleq (y_{un})_{u\in U, n\in N}$ denotes the request scheduling. A user u is admitted if and only if $\sum_{n\in N} y_{un} > 0$. Using these variables, we can formulate our problem as an *integer linear program (ILP)*, called the *Service Placement and Request Scheduling (SPRS) problem*:

$$\max \sum_{u\in U}\sum_{n\in N} y_{un} \tag{1a}$$

$$\text{s.t.} \sum_{n\in N} y_{un} \le 1, \qquad \forall u \in U, \tag{1b}$$

$$\sum_{l\in L} x_{ln} \le R, \qquad \forall n \in N, \tag{1c}$$

$$\sum_{u\in U_n}\sum_{n'\in N} y_{un'} \le K, \qquad \forall n \in N, \tag{1d}$$

$$\sum_{u\in U} y_{un} \le W, \qquad \forall n \in N, \tag{1e}$$

$$y_{un} \le a_{un}\cdot x_{l_u n}, \qquad \forall u \in U,\ n \in N, \tag{1f}$$

$$x_{ln}, y_{un} \in \{0,1\}, \qquad \forall l \in L, u \in U, n \in N. \tag{1g}$$

Objective (1a) maximizes the number of served users. Constraint (1b) ensures that each user is only counted once. Constraints (1c) and (1f) ensure that each edge cloud stores no more than R services and that an edge cloud can only serve a user if it is a candidate server and has the requested service. Constraint (1e) ensures that no more than W users are scheduled onto each edge cloud, and constraint (1d) ensures that each edge cloud admits (i.e., communicates over the wireless link with) no more than K users within its coverage, regardless of where they are scheduled. Finally, (1g) ensures that all the decision variables are indicators.

Remark: While our explicit objective is to maximize the number of served users, the formulation equivalently minimizes the cost in serving *all* the users, where users not served by edge clouds will be served by the backend cloud at a higher cost. The assumption is that serving each user at the edge provides a fixed cost saving, and the cost saving is the same for all the users. While the objective function (1a) gives equal weights to all y_{un}'s, we can add weights to reflect different cost savings for different users and/or edge clouds serving the users, and our solutions can be easily extended for this case.

III. Complexity Analysis

The SPRS problem (1) is related to but different from several known problems in the theory literature:

- SPRS differs from the *data placement problem (DPP)* [36], which generalizes the uncapacitated facility location problem. DPP only considers the resource of cache space that can be shared among requests for the same data object.
- SPRS differs from the *generalized assignment problem (GAP)* [37], which generalizes the knapsack and the multiple knapsack problems. In GAP, packing an item into a bin only consumes resource from the selected bin, whereas in SPRS, scheduling a user (i.e., user request) to an edge cloud not only consumes storage and computation resources at this edge cloud, but also consumes communication resources at the (possibly different) edge cloud covering the user.
- SPRS also differs from the *separable assignment problem (SAP)* [38], which allows general packing constraints for each bin that can model both non-sharable and sharable resources. As a special case, SAP includes the *distributed caching problem (DCP)* [38], which is about assigning content requests to caches subject to cache storage and bandwidth constraints. In DCP, requests for the same content can be served by the same content replica, but each request consumes a certain amount of bandwidth. Nevertheless, SAP (or DCP) requires the resource consumption to be limited to the selected bin (or cache), which differs from SPRS.

Intuitively, consuming both local and non-local resources makes SPRS harder than the above problems, which are already NP-hard. In the following, we will not only confirm this intuition, but also identify the root cause of hardness.

A. Special Cases

The three types of resource constraints, i.e., R-constraints (1c), K-constraints (1d), and W-constraints (1e), have different impacts on the complexity of the SPRS problem. To illustrate this point, we analyze the following special cases.

1) Case 1 – Removing K-constraints: If $K \ge |U|$, then the communication constraints (1d) are always satisfied, and can thus be removed from the ILP in (1). We will show that this special case is at least as hard as the *3-partition problem* and thus NP-hard.

Theorem III.1. The SPRS problem in (1) is NP-hard even without K-constraints (1d).

Proof. We prove the hardness by a reduction from the *3-partition problem* [39]. Given a set of $3k$ positive integers $S = \{t_1, \ldots, t_{3k}\}$ and $v = \frac{1}{k}\sum_{i=1}^{3k} t_i$, find subsets $S_1, \ldots, S_k$ of S of size 3 such that $\sum_{t_i\in S_j} t_i = v$ for all $j = 1, \ldots, k$.

We can reduce the 3-partition problem to our problem as follows. Suppose that there are $3k$ services, where service i has t_i users ($i = 1, \ldots, 3k$). In addition, there are k edge clouds, each with storage capacity $R = 3$, computation capacity $W = v$, and communication capacity $K = \sum_{i=1}^{3k} t_i$. Let $a_{un} \equiv 1$ for all the users and all the edge clouds. Then we claim that the 3-partition problem is feasible if and only if we can serve all the users, i.e., the optimal objective value of the constructed instance of SPRS equals $\sum_{i=1}^{3k} t_i$. First, given a feasible 3-partition $(S_j)_{j=1}^{k}$, placing the services corresponding to the integers in S_j in edge cloud j and scheduling all the users of these services to edge cloud j give a feasible solution to SPRS that serves all the users. Moreover, given a feasible solution to SPRS that serves all the users, grouping the integers corresponding to services placed on the same edge cloud gives a feasible 3-partition. As the 3-partition problem is NP-complete [39], the SPRS problem without K-constraints is NP-hard. □

2) Case 2 – Removing W-constraints: If $W \geq |U|$, then the computation constraints (1e) are always satisfied and can thus be removed from the ILP in (1). We will show that this special case is at least as hard as the *maximum coverage problem* and thus NP-hard.

Theorem III.2. The SPRS problem in (1) is NP-hard even without W-constraints (1e).

Proof. We reduce the maximum coverage problem to our problem. Given a positive integer I and a collection of sets $\mathcal{S} = \{S_1, \ldots, S_J\}$ $(I < J)$, the maximum coverage problem is to find a subset $\mathcal{S}' \subseteq \mathcal{S}$ of I sets such that the number of covered elements $|\bigcup_{S_i \in \mathcal{S}'} S_i|$ is maximized. Let $Q \triangleq \bigcup_{S_i \in \mathcal{S}} S_i$ be the ground set. Without loss of generality, we assume that $I < |Q|$ and $\max_{S_i \in \mathcal{S}} |S_i| < |Q|$ (otherwise the solution is trivial).

We construct an instance of the SPRS problem as follows. We construct one edge cloud n_e for each element $e \in Q$, and give each edge cloud a communication capacity $K = 1 + |Q| - I$, a storage capacity $R = 1$, and a computation capacity $W = |Q|(|Q| - I) + \sum_{i=1}^{J} |S_i|$. For each set $S_i \in \mathcal{S}$ $(i = 1, \ldots, J)$, we construct a service l_i. For each $e \in S_i$, we construct a user u with $l_u = l_i$ and $n_u = n_e$. Moreover, we construct $|Q| - I$ additional services $l_{J+1}, \ldots, l_{J+|Q|-I}$, each requested by one user in each cell. Let $a_{un} \equiv 1$. Then we claim that the optimal solution to the constructed instance of SPRS gives the optimal solution to the maximum coverage problem. This is because the optimal solution to SPRS must have selected services $l_{J+1}, \ldots, l_{J+|Q|-I}$ as they are requested from all the cells. After placing these services on any set of $|Q| - I$ edge clouds, the remaining edge clouds can only store $|N|R - |Q| + I = I$ more services from $\{l_1, \ldots, l_J\}$. Moreover, each edge cloud only has communication capacity for admitting one more user. This implies that given indices A of the services selected from $\{l_1, \ldots, l_J\}$ $(|A| \leq I)$, the number of served users, in addition to those requesting services $l_{J+1}, \ldots, l_{J+|Q|-I}$, equals $|\bigcup_{i \in A} S_i|$. Thus, if A^* is the set of service indices that maximizes the number of served users, then $\mathcal{S}' \triangleq \{S_i : i \in A^*\}$ is the optimal solution to the maximum coverage problem. □

3) Case 3 – Removing R-constraints: If $R \geq |L|$ (i.e., one edge cloud can store all the services), then the solution to x_{ln} is trivially $x_{ln} \equiv 1$ $(\forall l \in L$ and $n \in N)$. Under this service placement, the ILP in (1) is reduced to:

$$\max \sum_{u \in U} \sum_{n \in N} y_{un} \tag{2a}$$

$$\text{s.t.} \sum_{n \in N} y_{un} \leq 1, \qquad \forall u \in U, \tag{2b}$$

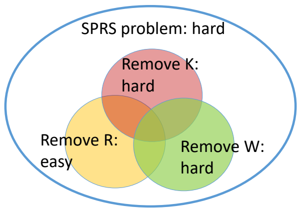


Fig. 2. Complexity of SPRS and its special cases.

$$\sum_{u \in U_n} \sum_{n' \in N} y_{un'} \leq K, \qquad \forall n \in N, \tag{2c}$$

$$\sum_{u \in U} y_{un} \leq W, \qquad \forall n \in N, \tag{2d}$$

$$y_{un} \leq a_{un}, \qquad \forall u \in U, n \in N, \tag{2e}$$

$$y_{un} \in \{0, 1\}, \qquad \forall u \in U, n \in N. \tag{2f}$$

In contrast to the previous cases, the special case in (2) is polynomial-time solvable. In fact, we will show later that the SPRS problem is polynomial-time solvable once the service placement is fixed (Section IV-A).

Lemma III.3. The SPRS problem without R-constraints, as is given in (2), is polynomial-time solvable.

Proof. As (2) is a special case of the request scheduling subproblem for a fixed service placement $x_{ln} \equiv 1$, the lemma is implied by the polynomial-time solution in Section IV-A. □

B. General Case

The hardness of the special cases (Theorems III.1 and III.2) implies that the SPRS problem is generally NP-hard.

Corollary III.4. The SPRS problem defined in (1) is NP-hard.

As illustrated in Fig. 2, besides establishing the hardness of the problem, our analysis also identifies the R-constraints as the cause of hardness. This insight motivates us to develop a two-step solution that separately addresses the request scheduling subproblem (focusing on K and W-constraints) and the service placement subproblem (focusing on R-constraints).

IV. Efficient Algorithms

The hardness of the optimal solution motivates our search for efficient suboptimal solutions. To this end, we develop a polynomial-time solution with a constant-factor approximation guarantee under certain conditions, and two efficient heuristics.

A. Optimal Algorithm for Request Scheduling

We will show that the hardness of the SPRS problem is due to the hardness of finding the optimal service placement, while the optimal request scheduling can be computed in polynomial time. The key is to note that under any given service placement, our problem can be converted to a maximum flow problem in an auxiliary graph.

Graph construction: Given a feasible service placement solution $\mathbf{x} \triangleq (x_{ln})_{l \in L, n \in N}$ satisfying constraint (1c), we construct an auxiliary graph $\mathcal{G}$ as illustrated in Fig. 3. The nodes in $\mathcal{G}$ consist of a source s, a destination d, a set of nodes U in 1-1 correspondence with the users, and two sets of nodes N_1 and N_2 each in 1-1 correspondence with the edge clouds. Node s is connected to each node in N_1 by a directed link of capacity K, and each node in N_2 is connected to node d by a directed link

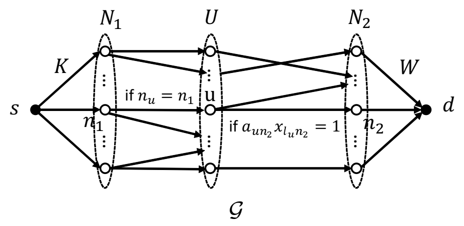


Fig. 3. Auxiliary graph $\mathcal{G}$ for a given service placement $\mathbf{x}$.

of capacity W. Moreover, each node $n_1 \in N_1$ is connected to each node $u \in U$ by a directed link of unit capacity if $n_u = n_1$ (i.e., user u is covered by edge cloud n_1), and each node $u \in U$ is connected to each node $n_2 \in N_2$ by a directed link of unit capacity if $a_{un_2}x_{l_u n_2} = 1$ (i.e., edge cloud n_2 is a candidate server for user u and has the requested service). Note that the topology of $\mathcal{G}$ depends on the service placement $\mathbf{x}$.

We will show that the request scheduling subproblem for a given $\mathbf{x}$ is equivalent to a maximum flow problem in $\mathcal{G}$.

Theorem IV.1. Given a service placement $\mathbf{x}$, the optimal value of (1) equals the maximum flow between s and d in the graph $\mathcal{G}$ defined in Fig. 3, and the optimal request scheduling is given by setting $y_{un} = 1$ if and only if link $(u, n) \in U \times N_2$ carries flow under the integral maximum flow from s to d.

Proof. First, by the *Integral Flow Theorem* [40], there exists an integral maximum flow between s and d as link capacities in $\mathcal{G}$ are all integral, and hence $\mathbf{y}$ is well-defined. Moreover, an s-to-d flow satisfies link capacities in $\mathcal{G}$ if and only if the corresponding $\mathbf{y}$ satisfies the constraints of (1). This is because by construction, the capacity constraints for links in $\{s\} \times N_1$ represent constraints (1d), those for links in $N_1 \times U$ represent constraints (1b), those for links in $U \times N_2$ represent constraints (1f), and those for links in $N_2 \times \{d\}$ represent constraints (1e). Note that constraints (1c) are already satisfied by the given $\mathbf{x}$, and constraints (1g) are satisfied by construction. Finally, under an integral solution, the value of s-to-d flow in $\mathcal{G}$ equals the value of the objective of (1). This is because as the links in $U \times N_2$ form a cut between s and d, the value of s-to-d flow equals the sum of flows across these links, which by definition equals the value of (1a). Therefore, the $\mathbf{y}$ constructed from the integral maximum s-to-d flow is the optimal solution to (1) under the given $\mathbf{x}$, and the optimal objective value equals the value of the maximum s-to-d flow. □

Algorithm: By Theorem IV.1, once the service placement is fixed, we can solve the request scheduling subproblem optimally. The solution, called *Optimal Request Scheduling (ORS)*, is shown in Algorithm 1. In computing the maximum s-to-d flow in $\mathcal{G}$ (line 3), we can leverage existing algorithms for computing the maximum flow in directed graphs. In particular, the Ford-Fulkerson algorithm [40] has guaranteed termination and optimality in this case. More importantly, this algorithm gives a maximum flow solution that is integral, i.e., only sending an integral amount of flow per link. The optimality of this algorithm is guaranteed by Theorem IV.1.

Corollary IV.2. Given a service placement $\mathbf{x}$, ORS maximizes the number of served users.

Complexity: It is easy to see that constructing $\mathcal{G}$ (line 2) takes $O(|N| \cdot |U|)$ time, dominated by the construction of links in $U \times N_2$. It is known that for integral link capacities, the Ford-Fulkerson algorithm has complexity $O(|E| \cdot F)$, where $|E|$ is the number of links and F is the maximum flow. In our case, $|E| = O(|N| \cdot |U|)$ and $F = O(|U|)$. Thus, computing the maximum flow (line 3) takes $O(|N| \cdot |U|^2)$ time. Therefore, the overall complexity of Algorithm 1 is $O(|N| \cdot |U|^2)$.

Algorithm 1: Optimal Request Scheduling (ORS)

```
input : Input parameters of (1) and service placement $\mathbf{x}$
output: Request scheduling $\mathbf{y}$
1 $\mathbf{y} \leftarrow \mathbf{0}$;
2 $\mathcal{G} \leftarrow$ auxiliary graph as in Fig. 3 for the service placement $\mathbf{x}$;
3 compute the integral maximum flow from $s$ to $d$ in $\mathcal{G}$;
4 foreach $(u, n) \in U \times N$ do
5     $y_{un} \leftarrow 1$ if link $(u, n)$ in $\mathcal{G}$ carries flow;
```

Observation: Combining the hardness of the overall SPRS problem (Corollary III.4) and the optimality of a polynomial-time solution to the request scheduling subproblem (Corollary IV.2) yields the following result.

Corollary IV.3. The service placement subproblem of the SPRS problem (1) is NP-hard.

B. Approximation Algorithm for Service Placement

Since the service placement subproblem is NP-hard, we focus on developing approximations for this subproblem. To this end, we show that under the optimal request scheduling, the optimization of service placement has certain properties that allow easy approximation. We introduce the following concepts from combinatorial optimization.

Definition 1 (Matroid [41])**.** A matroid $\mathbb{M}$ is a pair $(E, \mathcal{I})$, where E is a finite ground set and $\mathcal{I} \subseteq 2^E$ a non-empty collection of subsets of E, with the following properties:

- $\forall A \subset B \subseteq E$, if $B \in \mathcal{I}$, then $A \in \mathcal{I}$;
- $\forall A,\ B \in \mathcal{I}$ with $|B| > |A|$, $\exists x \in B \setminus A$ such that $A \cup \{x\} \in \mathcal{I}$.

Definition 2 (Monotone submodular function [41])**.** Given a finite ground set E and a function $f : 2^E \to \mathbb{R}$,

- f is monotone if $\forall A \subset B \subseteq E$, $f(A) \leq f(B)$;
- f is submodular if $\forall A \subset B \subseteq E$ and $e \in E \setminus B$, $f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B)$.

If our objective function is monotone submodular and our constraint forms a matroid, then the *greedy algorithm*, that selects one element at a time such that each selection maximizes the objective function, achieves $1/2$-approximation.

Theorem IV.4 ([42])**.** Consider the problem of maximizing a set function $f : 2^E \to \mathbb{R}$ over a collection of sets $\mathcal{I} \subseteq 2^E$. Let f^* denote the optimal value and f^g the value achieved by the greedy algorithm. If $\mathbb{M} = (E, \mathcal{I})$ is a matroid and f is monotone and submodular, then $f^g \geq f^*/2$.

We will show that under certain conditions, the service placement subproblem is maximizing a monotone submodular set function under a matroid constraint. To this end, we rewrite our problem as a set optimization. Let $S(\mathbf{x}) \triangleq \{(l, n) \in L \times N : x_{ln} = 1\}$ denote the set of single-service placements according to $\mathbf{x}$, where $(l, n) \in S(\mathbf{x})$ means to place a replica of service l at edge cloud n. Let $\Omega(S(\mathbf{x}))$ denote the optimal objective value of (1) for a given $\mathbf{x}$. Simply writing $S(\mathbf{x})$ as S, we can rewrite the service placement subproblem as:

$$\max \Omega(S) \tag{3a}$$
$$\text{s.t. } |S \cap S_n| \le R, \qquad \forall n \in N, \tag{3b}$$
$$S \subseteq L \times N, \tag{3c}$$

where $S_n \triangleq L \times \{n\}$ is the set of all possible single-service placements at edge cloud n. We have the following observations:

- *Matroid constraint:* Let $\mathcal{I}$ be the collection of all the S's satisfying constraints (3b, 3c). Then it is easy to verify that $\mathbb{M} = (L \times N, \mathcal{I})$ is a matroid. This is known as the *partition matroid* as $\{S_n\}_{n\in N}$ is a partition of the ground set $L \times N$.
- *Monotone submodular objective function:* We show that the objective function (3a) has the following properties.

Lemma IV.5. Function $\Omega(S)$ is monotone and submodular for any feasible S if (i) $R = 1$ *or* (ii) $W \ge |U|$.

Proof. First, we note that adding an element (l, n) to S corresponds to adding a set of links $\mathcal{L}_{ln} \triangleq \{(u, n) : a_{un} = 1, l_u = l\}$ between nodes U and N_2 in $\mathcal{G}$ (Fig. 3). Since adding links can only increase the maximum flow, adding elements to S can only increase $\Omega(S)$, i.e., $\Omega(S)$ is monotone increasing.

To prove submodularity of $\Omega(S)$, consider any $S_1 \subseteq S_2$ and $(l, n) \notin S_2$. Let π_i $(i = 1, 2)$ be an optimal schedule under service placement $\{(l, n)\} \cup S_i$ that minimizes the number of users served by (l, n), and $\mathcal{U}((l, n)|S_i)$ be the set of users served by (l, n) under π_i. It suffices to show that $|\mathcal{U}((l, n)|S_2)| \le |\mathcal{U}((l, n)|S_1)|$. For any $u \in \mathcal{U}((l, n)|S_2)$, under π_1, u must be (1) served by (l, n), (2) served by (l, n') for $n' \ne n$, or (3) not served. In case (2), modifying π_2 to schedule u to (l, n') will give an optimal schedule under $\{(l, n)\} \cup S_2$ that schedules fewer users to (l, n) than π_2, which is a contradiction. The modified schedule must be feasible if $R = 1$ or $W \ge |U|$, as under these conditions, the additional replicas in $S_2 \setminus S_1$ will not restrict the users that (l, n') can serve. In case (3), there must exist u' $(u' \ne u)$ served by (l', n') under π_1, that blocks u from being served (due to K/W-constraints). If $(l', n') \ne (l, n)$, then modifying π_2 to schedule u' to (l', n') instead of u to (l, n) will give an optimal schedule under $\{(l, n)\} \cup S_2$ that schedules fewer users to (l, n) than π_2, which is again a contradiction. Thus, each $u \in \mathcal{U}((l, n)|S_2)$ must be either served by (l, n) or in one-one correspondence with another user served by (l, n) under π_1, which proves $|\mathcal{U}((l, n)|S_2)| \le |\mathcal{U}((l, n)|S_1)|$. □

Counterexample: We note that $\Omega(S)$ is *not* submodular in general. Consider the case in Fig. 4. Given the placement (l_1, n_2), adding (l_1, n_1) does not increase the number of served users. However, given the placement $\{(l_1, n_2), (l_2, n_2)\}$, adding (l_1, n_1) helps to serve both users instead of only one.

Algorithm: The set function formulation (3) inspires us to apply existing algorithms for such problems. In particular, applying the generic greedy algorithm yields Algorithm 2, referred to as *Greedy Service Placement with Optimal Request Scheduling (GSP-ORS)*. Starting from an empty placement, the algorithm iteratively places one single service at a time

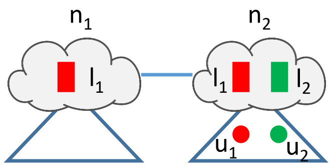


Fig. 4. Example for non-submodularity of $\Omega(S)$ ($R = 2$, $K = 2$, $W = 1$, $a_{un} \equiv 1$); colors indicate the types of services.

Algorithm 2: Greedy Service Placement with Optimal Request Scheduling (GSP-ORS)

```
input : Input parameters of (1)
output: Service placement x and request scheduling y
1  $S \leftarrow \emptyset$;
2  $\omega^* \leftarrow 0$;
3  while $\exists n \in N$ with $|S \cap S_n| < R$ do
4      $(l^*, n^*) \leftarrow \emptyset$;
5      foreach $(l, n) \notin S$ such that $|S \cap S_n| < R$ do
6          $\omega \leftarrow \Omega(S \cup \{(l, n)\})$;
7          if $\omega > \omega^*$ then
8              $(l^*, n^*) \leftarrow (l, n)$;
9              $\omega^* \leftarrow \omega$;
10     if $(l^*, n^*) \ne \emptyset$ then
11         $S \leftarrow S \cup \{(l^*, n^*)\}$;
12     else
13         break;
14 convert $S$ to its vector representation $\mathbf{x}$;
15 compute $\mathbf{y}$ by ORS (Algorithm 1) for input $\mathbf{x}$;
```

until all the edge clouds are full (lines 3–13), such that each single-service placement maximizes the objective value (lines 5–9). Note that each evaluation of the objective function $\Omega(S \cup \{(l, n)\})$ (line 6) requires an invocation of ORS (where the objective value is given by the maximum flow computed in line 3 of Algorithm 1).

Complexity: There are $O(|N|R)$ iterations in Algorithm 2, within each of which the algorithm considers $O(|L| \cdot |N|)$ candidate single-service placements and evaluates the objective function for each candidate placement by solving an $O(|N| \cdot |U|^2)$-complexity request scheduling subproblem. Therefore, the overall complexity of Algorithm 2 is $O(|N|^3|U|^2|L|R)$.

Approximation guarantee: When our objective is monotone and submodular (Lemma IV.5), we can apply Theorem IV.4 to guarantee the following.

Corollary IV.6. If $R = 1$ *or* $W \ge |U|$, GSP-ORS (Algorithm 2) achieves an approximation ratio of $1/2$, i.e., the number of users served under this solution is at least half of the maximum number of users served according to SPRS (1).

C. Heuristics

Although GSP-ORS has a polynomial complexity of $O(|N|^3|U|^2|L|R)$, the order of this polynomial is rather high, which makes the algorithm slow for large systems. To further simplify computation, we develop two heuristics.

1) Greedy Heuristic: This is a variation of GSP-ORS that performs both the service placement and the request scheduling greedily. Specifically, we replace the optimal scheduling in line 6 of Algorithm 2 by scheduling as many users as possible, *without rescheduling any previously scheduled users.*

To achieve this, we introduce the following variables: $\widetilde{W}_n$ denoting the residual computation capacity at edge cloud n, $\widetilde{K}_n$ denoting the residual communication capacity at edge

Algorithm 3: Greedy Service Placement with Greedy Request Scheduling (GSP-GRS)

```
input : Input parameters of (1)
output: Service placement $\mathbf{x}$ and request scheduling $\mathbf{y}$
1  $\mathbf{x} \leftarrow \mathbf{0}$;
2  $\mathbf{y} \leftarrow \mathbf{0}$;
3  foreach $n \in N$ do
4      $\widetilde{W}_n \leftarrow W$;
5      $\widetilde{K}_n \leftarrow K$;
6      foreach $l \in L$ do
7          $\widetilde{U}_{ln} \leftarrow U_{ln}$;
8  while $\Phi(\mathbf{x}) \neq \emptyset$ do
9      $(l^*, n^*) \leftarrow \arg\max_{(l,n)\in\Phi(\mathbf{x})} \min(\widetilde{W}_n, \sum_{n'\in N} \min(|\widetilde{U}_{ln'} \cap V_n|, \widetilde{K}_{n'}))$;
10     $x_{l^*n^*} \leftarrow 1$;
11     $o^* \leftarrow \min(\widetilde{W}_{n^*}, \sum_{n\in N} \min(|\widetilde{U}_{l^*n} \cap V_{n^*}|, \widetilde{K}_n))$;
12     if $o^* = 0$ then
13         break;
14     $\widetilde{W}_{n^*} \leftarrow \widetilde{W}_{n^*} - o^*$;
15     foreach $n \in N$ do
16         $o \leftarrow \min(o^*, \min(|\widetilde{U}_{l^*n} \cap V_{n^*}|, \widetilde{K}_n))$;
17         $\widetilde{K}_n \leftarrow \widetilde{K}_n - o$;
18         randomly select a set $U' \subseteq \widetilde{U}_{l^*n} \cap V_{n^*}$ with $|U'| = o$;
19         foreach $u \in U'$ do
20             $y_{un^*} \leftarrow 1$;
21         $\widetilde{U}_{l^*n} \leftarrow \widetilde{U}_{l^*n} \setminus U'$;
22         $o^* \leftarrow o^* - o$;
23         if $o^* = 0$ then
24             break;
```

cloud n, and $\widetilde{U}_{ln}$ denoting the set of unserved users covered by edge cloud n that request service l. Let $U_{ln} \triangleq \{u \in U : n_u = n,\ l_u = l\}$ denote all the users covered by edge cloud n that request service l, and $V_n \triangleq \{u \in U : a_{un} = 1\}$ denote all the users that can be served by edge cloud n. We develop a greedy heuristic referred to as *Greedy Service Placement with Greedy Request Scheduling (GSP-GRS)*, shown in Algorithm 3. GSP-GRS is similar to GSP-ORS in that it still places services iteratively (lines 8–24). The difference is that within each iteration, it places the additional service that serves the maximum number of new (i.e., previously unscheduled) users without rescheduling any of the existing users, by only using the residual capacities $\widetilde{W}_n$ and $\widetilde{K}_n$ to schedule the unserved users $\widetilde{U}_{ln}$.

Specifically, we note that placing an additional service l at edge cloud n (assuming l was not placed at n) serves at most

$$\min(\widetilde{W}_n, \sum_{n'\in N} \min(|\widetilde{U}_{ln'} \cap V_n|,\ \widetilde{K}_{n'})) \tag{4}$$

new users without rescheduling the existing users. Let

$$\Phi(\mathbf{x}) \triangleq \{(l,n) \in L \times N :\ x_{ln} = 0, \sum_{l'\in L} x_{l'n} < R\} \tag{5}$$

denote the set of feasible placements of one additional service. GSP-GRS selects the placement in $\Phi(\mathbf{x})$ that maximizes (4) (line 9). After placing the selected service (line 10), it computes the number of newly served users o^* (line 11). If $o^* = 0$, the algorithm stops (line 13). Otherwise, it sequentially goes through the edge clouds to schedule eligible unserved users and updates the residual capacities (lines 14–24).

Complexity: The **while** loop is repeated $O(|N|R)$ times, and the computation within the loop is bottlenecked by the service placement selection (line 9) which takes $O(|L||N||U|)$ time, while the remaining steps (lines 10-24) take $O(|U|)$ time. Therefore, the overall complexity of Algorithm 3 is $O(|N|^2|U||L|R)$. Compared with Algorithm 2, Algorithm 3 reduces the complexity by a factor of $O(|N||U|)$.

2) LP Relaxation with Rounding: The ILP formulation of the SPRS problem (1) allows us to apply linear program (LP) relaxation. This method first computes a fractional solution to (1) by relaxing the integer constraints (1g) to linear constraints $x_{ln}, y_{un} \in [0,1]$ ($\forall l \in L, u \in U, n \in N$), and then rounds the solution to integers as follows. For each edge cloud $n \in N$, we sort the services into descending order of the fractional x_{ln}'s ($\forall l \in L$), and place the top R services in edge cloud n; for each user $u \in U$, we sort the edge clouds into descending order of the fractional y_{un}'s ($\forall n \in N$), and schedule u to the first available edge cloud (if any), subject to the constraints in (1).

Complexity: The dominating step is to solve the LP relaxation of (1). This LP has $O(|N|(|L| + |U|))$ variables and $O(|N|(|L| + |U|))$ constraints, and thus can be solved in $O(|N|^{7.5}(|L| + |U|)^{7.5})$ time by *Karmarkar's algorithm* [44]. Thus, the complexity of LP relaxation with rounding is $O(|N|^{7.5}(|L| + |U|)^{7.5})$. Although in theory this is slower than GSP-ORS, in practice it can be faster (see Table II) by leveraging existing efficient LP solvers.

V. Extension to Heterogeneous Case

In general, different edge clouds can have different capacities, and different services can require different amounts of resources. We now characterize the impact of heterogeneity on the problem complexity and the proposed solutions.

A. Generalized Optimization

To model heterogeneity among the edge clouds, we generalize the parameters K, W, and R to K_n, W_n, and R_n ($n \in N$), respectively, to denote the communication/computation/storage capacity of edge cloud n, which can be different for different edge clouds. Similarly, to model heterogeneity among the services, we introduce new parameters κ_l, ω_l, and r_l ($l \in L$) to denote the communication/computation/storage requirement of service l, where the communication/computation requirement is per request, and the storage requirement is per placement (of a service replica). Using these parameters, we generalize the SPRS problem (1) to the following ILP, referred to as the *Generalized SPRS problem*:

$$\max \sum_{u\in U} \sum_{n\in N} y_{un} \tag{6a}$$

$$\text{s.t.} \sum_{n\in N} y_{un} \leq 1, \qquad \forall u \in U, \tag{6b}$$

$$\sum_{l\in L} x_{ln} \cdot r_l \leq R_n, \qquad \forall n \in N, \tag{6c}$$

$$\sum_{u\in U_n} (\sum_{n'\in N} y_{un'}) \cdot \kappa_{l_u} \leq K_n, \qquad \forall n \in N, \tag{6d}$$

$$\sum_{u\in U} y_{un}\cdot \omega_{l_u} \le W_n, \qquad \forall n\in N, \tag{6e}$$

$$y_{un} \le a_{un}\cdot x_{l_u n}, \qquad \forall u\in U,\ n\in N, \tag{6f}$$

$$x_{ln}, y_{un} \in \{0,1\}, \qquad \forall l\in L, u\in U, n\in N. \tag{6g}$$

B. Complexity Analysis

As (6) is a generalization of (1), which is NP-hard (Corollary III.4), the Generalized SPRS problem is NP-hard. Meanwhile, we show that while the request scheduling subproblem is polynomial-time solvable in the homogeneous case (Corollary IV.2), it becomes NP-hard in the heterogeneous case.

Theorem V.1. Even if the service placement $\mathbf{x}$ is given, the request scheduling subproblem of the Generalized SPRS problem (6) is still NP-hard.

Sketch of proof. We prove the result by constructing an instance of the request scheduling subproblem of (6) that is equivalent to the *partition problem*, which is known to be NP-complete. We refer to [45] for details. □

C. Generalized Algorithms

While GSP-ORS (Algorithm 2) cannot be extended to the heterogeneous case due to the hardness of optimal request scheduling, both GSP-GRS (Algorithm 3) and the LP relaxation method (Section IV-C2) can be extended to the heterogeneous case. We refer to [45] for the details.

VI. Performance Evaluation

We have evaluated the performance of the proposed algorithms against benchmarks via both synthetic and trace-driven simulations. Due to space limitation, we only present the results of trace-driven simulations, and refer to [45] for additional results from synthetic simulations.

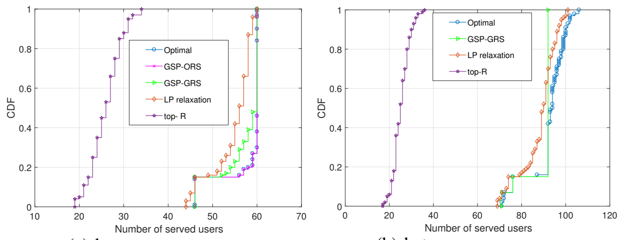
A. Benchmarks

As benchmarks for our algorithms, we evaluate: (i) the optimal solution obtained by solving (1) and (6) using an ILP solver, and (ii) a baseline solution that first places the top-R (homogeneous case) or top-R_n (heterogeneous case) most popular services in each edge cloud, and then schedules as many users as possible via ORS (homogeneous case) or greedy scheduling (heterogeneous case). The service placement strategy in the baseline is the same as the *offline static policy* in [28]. Here the service popularity for an edge cloud is computed based on *all possible requests* that can be scheduled to it, i.e., $\{u \in U:\ a_{un}=1\}$. This baseline, referred to as 'top-R', is used to evaluate the performance of optimizing service placement and request scheduling *separately* (as it ignores how requests will be scheduled when placing services).

B. Simulation Setting

We extract user and edge cloud locations from real mobility traces and cell tower locations. We use the taxi cab traces from [46], from which we extract the traces of 280 users (i.e., taxi cabs) over a 100-minute period with location updates every minute[3]. We quantize the user

[3]We filter out inactive nodes with no update for at least 5 minutes and regulate the update intervals through linear interpolation.



(a) homogeneous case (b) heterogeneous case

Fig. 5. Performance comparison in the basic setting.

locations into Voronoi cells based on cell tower locations in the geographical area covered by the traces, obtained from `http://www.antennasearch.com`, and find that at most 6 cells are occupied in any 1-minute slot. To simulate nontrivial scenarios, we set $|N| = 6$ and assume that only the occupied cells can host services[4]. Unless stated otherwise, we generate a request for each user in each slot according to the Zipf distribution with exponent α. We set $|L| = 1000$ and $\alpha = 0.6$. In the homogeneous case, we set $K = 15, R = 5, W = 10$; in the heterogeneous case, we uniformly draw $K_n \in [10, 15]$, $R_n \in [1, 5]$, $W_n \in [5, 10]$ for each $n \in N$, and $\kappa_l, r_l, \omega_l \in [0.1, 1]$ for each $l \in L$. We set $a_{un} \equiv 1$ ($\forall u \in U,\ n \in N$). Similar results have been observed under other parameter settings.

All the algorithms are implemented in MATLAB R-2017a, and evaluated on a machine with Mac OS 64 bits, 2.8 GHz Intel Core i5 Processor, and 8 GB 1600 MHz DDR3 memory.

C. Evaluation Results

1) Performance Comparison: Fig. 5 shows the cumulative distribution function (CDF) of the number of served users over the 100 slots. In the homogeneous case (Fig. 5(a)), the average number of served users is 57.68 for both the optimal solution and GSP-ORS, 56.85 for GSP-GRS, 54.73 for LP relaxation (with rounding), and 26.04 for top-R; in the heterogeneous case (Fig. 5(b)), this number is 92.22 for the optimal solution, 89.25 for (the generalized) GSP-GRS, 88.19 for LP relaxation, and 25.51 for top-R.

In both cases, there is a 2–3-fold gap between the baseline solution (top-R) that optimizes service placement and request scheduling separately, and the other solutions that consider the two problems jointly, which signals the importance of joint service placement and request scheduling. In this setting, the proposed algorithms (GSP-ORS, GSP-GRS, and LP relaxation) all achieve near-optimal performance, where the number of served users is within 5% of the optimal, and GSP-ORS is exactly optimal in the homogeneous case (Fig. 5(a)).

Meanwhile, these algorithms have very different average running times as presented in Table II. While the optimal solution is very slow due to the NP-hardness of the problem, GSP-ORS is also slow due to a large number of calls to ORS, and the other solutions are fast. In particular, GSP-GRS provides a good tradeoff between performance and complexity, serving a near-optimal number of users with very low running time (~ 50 times faster than LP relaxation). We note that the presented running times are based on our initial

[4]There are many more (> 280) cell towers in the area covered by the traces, and thus the resource provisioning problem will become trivial if edge clouds are deployed at all the cells.

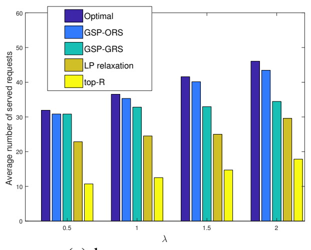
TABLE II
RUNNING TIME FOR PROPOSED ALGORITHMS VS. OPTIMAL SOLUTION

Algorithm	Average Running time
Optimal via brute-force search[5]	8.7357e+75 sec
GSP-ORS	535.9117 sec
GSP-GRS	0.0188 sec
LP relaxation with rounding	0.9852 sec
top-R (with best-effort scheduling)	0.0277 sec

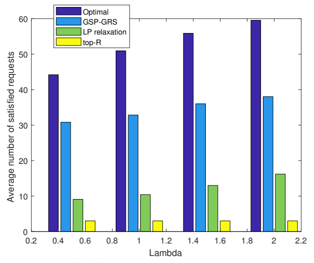
implementation in Matlab, and can be improved via more efficient implementation methods.

2) Stress Test: To stress-test the proposed solutions, we repeat the above simulations under increasing loads. We tune the load by generating a Poisson(λ) number of requests per user in each slot, and vary the parameter λ (average number of requests per user per slot). Each request is independently drawn from the Zipf distribution with exponent α. To speed up the simulation, we reduce the number of users to 50 (randomly selected from all the users), and the number of edge clouds to 5 (as at most 5 cells are occupied by the selected users). The other parameters are the same as before.

Fig. 6 shows the average number of served requests under each value of λ. In the homogeneous case (Fig. 6(a)), as λ increases, GSP-ORS starts to show notably better performance than the heuristics. This result shows that besides having guaranteed approximation in the worst case, GSP-ORS also achieves superior average performance. Among the heuristics (GSP-GRS, LP relaxation, and top-R), GSP-GRS performs the best. In the heterogeneous case (Fig. 6(b)), the same comparison is observed. However, the best-performing heuristic (GSP-GRS) still has a substantial optimality gap, indicating room of improvement for more sophisticated algorithms.



(a) homogeneous case



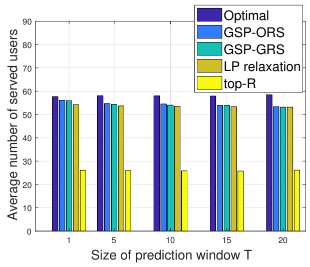
(b) heterogeneous case

Fig. 6. Performance comparison under increasing loads.

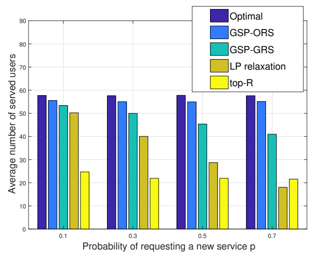
3) Sensitivity Analysis: To apply our solutions in an online setting, we need to predict user demands (i.e., locations and requested services). To understand the sensitivity of our solutions with respect to (wrt) prediction errors, we apply the proposed algorithms by first placing services according to each of these algorithms based on the predicted user demands, and then scheduling user requests based on the placed services and the actual user demands. In these simulations, we only generate one request per user in each slot. Below we only show the results for the homogeneous case due to space limitation.

Sensitivity to errors in user locations: We predict the user locations at time t based on their locations at time $t - T$.

[5]This is estimated by multiplying the number of possible service placements with the time to compute the optimal request scheduling for each placement, estimated by the running time of ORS (the heterogeneous case is even harder). This is for a fair comparison with the other algorithms which are implemented as Matlab scripts. In the simulations, we used a commercial-grade ILP solver (Matlab `intlinprog`) to compute the optimal solution.



(a) errors in predicted user locations



(b) errors in predicted user requests

Fig. 7. Sensitivity analysis.

Fig. 7(a) shows the average number of served users as we vary T, where the optimal solution is computed based on current user locations (no prediction error), while the rest are based on predicted user locations. We see that the algorithms are robust against errors in user locations. We have verified that there are substantial errors in the predicted user locations and the error rate does increase with T (from 30% of error for $T = 1$ to 87% of error for $T = 20$). Nevertheless, as we allow requests to be served by non-local edge clouds, the dominating factor for the service placement subproblem is the types of requested services, not where the users are located.

Sensitivity to errors in user requests: We perform a similar experiment by predicting the types of requested services. We assume that in each slot, each user requests a new service (drawn independently from the Zipf distribution with exponent α) with probability p, and requests the same service as in the previous slot with probability $1 - p$. The types of services requested at time t are predicted based on their values at time $t - 1$. Fig. 7(b) shows the average number of served users wrt p, where the optimal solution is based on current requests (no error) and the rest are based on predicted requests. We see that the performance of GSP-GRS and LP relaxation degrades quickly as p increases. The reason is that as the users change their interests more frequently, services placed to serve requests in the previous slot become less useful for the current slot. The performance of top-R stays roughly the same, as it is only based on service popularity, which remains the same on the average (as requests are drawn from the same distribution). Interestingly, we observe that GSP-ORS is highly robust against prediction errors in service types.

VII. CONCLUSION

We have studied the problem of joint service placement and request scheduling in mobile edge computing systems under communication, computation, and storage constraints. Through a thorough complexity analysis, we not only prove the NP-hardness of the problem, but also identify the root cause of hardness. We further propose a polynomial-time algorithm with approximation guarantee for the homogeneous case, and efficient heuristics for the general case. Our trace-driven evaluations show that the proposed solutions, especially the algorithm with approximation guarantee, can achieve near-optimal performance while significantly reducing the running time compared to the optimal solution.

ACKNOWLEDGMENT

The authors would like to thank Konstantinos Poularakis for the insightfully constructed counterexample on the submodularity of the proposed objective function.

References

[1] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, March 2017.

[2] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutotials*, vol. 19, no. 4, pp. 2322–2358, August 2017.

[3] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds," in *IFIP Networking*, May 2015.

[4] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta, J. Boleng, and K. Ha, "The role of cloudlets in hostile environments," *IEEE Pervasive Computing*, vol. 12, no. 4, pp. 40–49, October 2013.

[5] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *The First Edition of the MCC Workshop on Mobile Cloud Computing*, August 2012, pp. 13–16.

[6] T. Taleb and A. Ksentini, "Follow me cloud: Interworking federated clouds and distributed mobile networks," *IEEE Network*, vol. 27, no. 5, pp. 12–19, September 2013.

[7] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *accepted to IEEE Transactions on Parallel and Distributed Systems*, August 2016.

[8] K. Ha, Y. Abe, Z. Chen, W. He, B. Amos, P. Pillai, and M. Satyanarayanan, "Adaptive VM handoff across cloudlets," Technical Report CMU-CS-15-113, June 2015. [Online]. Available: https://www.cs.cmu.edu/~satya/docdir/CMU-CS-15-113.pdf

[9] K. Ha, Y. Abe, T. Eiszler, Z. Chen, W. Hu, B. Amos, R. Upadhyaya, P. Pillai, and M. Satyanarayanan, "You can teach elephants to dance: Agile VM handoff for edge computing," in *ACM/IEEE Symposium on Edge Computing (SEC)*, October 2017.

[10] L. Ma, S. Yi, and Q. Li, "Efficient service hadoff across edge servers via docker container migration," in *ACM/IEEE Symposium on Edge Computing (SEC)*, October 2017.

[11] A. Machen, S. Wang, K. K. Leung, B. Ko, and T. Salonidis, "Live service migration in mobile edge clouds," *to appear in IEEE Wireless Communications*, 2017.

[12] "Open Edge Computing." [Online]. Available: http://openedgecomputing.org/

[13] "OpenFog Consortium." [Online]. Available: https://www.openfogconsortium.org/

[14] "ETSI ISG on Multi-access Edge Computing (MEC)." [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing

[15] A. Ksentini, T. Taleb, and M. Chen, "A Markov decision process-based service migration procedure for Follow Me cloud," in *IEEE ICC*, June 2014.

[16] S. Wang, R. Urgaonkar, T. He, M. Zafer, K. Chan, and K. K. Leung, "Mobility-induced service migration in mobile micro-clouds," in *IEEE MILCOM*, October 2014.

[17] T. Taleb, A. Ksentini, and P. Frangoudis, "Follow-me cloud: When cloud services follow mobile users," *accepted to IEEE Transactions on Cloud Computing*, February 2016.

[18] L. Wang, L. Jiao, J. Li, and M. Muhlhauser, "Online resource allocation for arbitrary user mobility in distributed edge clouds," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 1281–1290.

[19] "Mobile-Edge Computing (MEC); Service Scenarios," ETSI GS MEC-IEG 004 v1.1.1, November 2015. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/MEC-IEG/001_099/004/01.01.01_60/gs_MEC-IEG004v010101p.pdf

[20] M. Jia, J. Cao, and W. Liang, "Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks," *IEEE Transactions on Cloud Computing*, vol. PP, no. 99, pp. 1–1, 2015.

[21] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo, "Efficient algorithms for capacitated cloudlet placements," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 10, pp. 2866–2880, Oct 2016.

[22] A. Ceselli, M. Premoli, and S. Secci, "Mobile edge cloud network design optimization," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1818–1831, June 2017.

[23] H. Tan, Z. Han, X.-Y. Li, and F. Lau, "Online job dispatching and scheduling in edge-clouds," in *IEEE INFOCOM*, May 2017.

[24] S. Wang, M. Zafer, and K. K. Leung, "Online placement of multi-component applications in edge computing environments," *IEEE Access*, vol. 5, pp. 2514–2533, 2017.

[25] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *IEEE INFOCOM 2016*, April 2016, pp. 1–9.

[26] G. Dán and N. Carlsson, "Dynamic content allocation for cloud-assisted service of periodic workloads," in *IEEE INFOCOM*, April 2014.

[27] S. Shukla, O. Bhardwaj, A. A. Abouzeid, T. Salonidis, and T. He, "Hold'em caching: Proactive retention-aware caching with multi-path routing for wireless edge networks," in *ACM Mobihoc*, July 2017.

[28] I.-H. Hou, T. Zhao, S. Wang, and K. Chan, "Asymptotically optimal algorithm for online reconfiguration of edge-clouds," in *Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, ser. MobiHoc '16, New York, NY, USA, 2016, pp. 291–300.

[29] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *IEEE INFOCOM*, April 2010.

[30] M. Dehghan, B. Jiang, A. Seetharam, T. He, T. Salonidis, J. Kurose, D. Towsley, and R. Sitaraman, "On the complexity of optimal request routing and content caching in heterogeneous cache networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1635–1648, June 2017.

[31] M. Barcelo, J. Llorca, A. M. Tulino, and N. Raman, "The cloud service distribution problem in distributed cloud networks," in *IEEE ICC*, June 2015.

[32] J. Llorca, A. M. Tulino, A. Sforza, and C. Sterle, "Optimal content distribution and multi-resource allocation in software defined virtual cdns," in *AIRO ODS*, September 2017.

[33] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, and K. K. Leung, "Dynamic service migration and workload scheduling in edge-clouds," *Performance Evaluation*, vol. 91, pp. 205–228, October 2015.

[34] H. Feng, J. Llorca, A. M. Tulino, D. Raz, and A. F. Molisch, "Approximation algorithms for the nfv service distribution problem," in *IEEE INFOCOM*, April 2017.

[35] Y. Rochman, H. Levy, and E. Brosh, "Resource placement and assignment in distributed network topologies," in *IEEE INFOCOM*, April 2013.

[36] I. Baev, R. Rajaraman, and C. Swamy, "Approximation algorithms for data placement problems," *SIAM Journal on Computing*, vol. 38, no. 4, pp. 1411–1429, 2008.

[37] D. B. Shmoys and Éva Tardos, "An approximation algorithm for the generalized assignment problem," *Mathematical Programming*, vol. 62, no. 1-3, pp. 461–474, February 1993.

[38] L. Fleischer, M. X. Goemans, V. S. Mirrokni, and M. Sviridenko, "Tight approximation algorithms for maximum general assignment problems," in *ACM-SIAM SODA*, January 2006.

[39] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, 1990.

[40] B. Korte and J. Vygen, "Network flows," in *Combinatorial Optimization*. Berlin, Germany: Springer, 2000, ch. 8, pp. 153–184.

[41] J. Lee, *A First Course in Combinatorial Optimization*. Cambridge University Press, 2004.

[42] M. Fisher, G. Nemhauser, and L. Wolsey, "An analysis of approximations for maximizing submodular set functions – II," *Math. Prog. Study*, vol. 8, pp. 73–87, 1978.

[43] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák, "Maximizing a monotone submodular function subject to a matroid constraint," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1740–1766, 2011.

[44] G. Strang, "Karmarkar's algorithm and its place in applied mathematics," *The Mathematical Intelligencer*, 1987.

[45] T. He, H. Khamfroush, S. Wang, T. L. Porta, and S. Stein, "It's Hard to Share: Joint Service Placement and Request Scheduling in Edge Clouds with Sharable and Non-sharable Resources," Technical Report, December 2017. [Online]. Available: https://1drv.ms/b/s!Av2FAzDDqJorbO61jkJMD9wGHzs

[46] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "CRAWDAD dataset epfl/mobility (v. 2009-02-24)," February 2009. [Online]. Available: http://crawdad.org/epfl/mobility/20090224