It's Hard to be Social: Joint Service Placement and Request Scheduling for Social Edge Applications

Ting He*, Hana Khamfroush*, Shiqiang Wang†, Tom La Porta*, and Sebastian Stein‡
*Pennsylvania State University, University Park, PA, USA. Email: {tzh58,hxk5299,tlp}@cse.psu.edu
†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—**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 make a simplifying assumption that resource demands of different users are additive, this assumption does not hold for social applications (e.g., social virtual reality), where users collaborating on a task are interested in services (e.g., scene updates) based on the same set of data/code. Meanwhile, serving each user request also consumes non-negligible additive resources (e.g., CPU, bandwidth). We study the optimal provisioning of such applications by jointly placing services and scheduling requests under both additive (communication, computation) and sub-additive (storage) resource constraints. 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, all achieving near-optimal performance in our trace-driven evaluations.**

I. Introduction

Edge computing (a.k.a. mobile edge computing) has been one of the fastest-arising 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, 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* [1] (a.k.a. *cloudlets*, *micro clouds*, *fog*, and *follow me cloud*). Edge computing lets users exploit the power of cloud computing without incurring the high latency in communicating to remote clouds.

A major limitation in edge computing is that compared with traditional data centers, each edge cloud is much more limited in resources. With the help of virtualization techniques such as virtual machines (VM) and Docker containers, the Open Edge Computing community [2] is developing open and globally standardized resource provisioning mechanisms, such that edge clouds within the same geographical region will form a shared resource pool. The research challenge is then to optimally allocate resources from this pool to competing demands.

While having received significant attention in recent years, existing solutions [3], [4], [1], [5], [6] mostly assume *additive* resource demands, i.e., the total resource demand on an edge cloud is the sum of all the demands scheduled to it. While this assumption holds for single-user applications where each user is served by a dedicated application instance (encapsulated in a VM/container), it fails to characterize the resource demands of social applications (e.g., social virtual reality and gaming). In social applications, a group of users collaborating on the same task need to access the same data/code relevant to the task (e.g., scenes and object recognition for the scenes), and therefore the storage demands of these users are *sub-additive*, i.e., the total storage demand on an edge cloud may be smaller than the sum of individual storage demands due to shared data/code. Meanwhile, serving each user request (e.g., update the scene) also consumes a non-negligible amount of additive resources such as CPU and bandwidth (assuming unicasts). The same holds for analytics applications (e.g., video analytics), where multiple users interested in the same analytics can share the same copy of data/code, while individual analytics requests compete for CPU and bandwidth. The fundamental problem in provisioning such applications from the network edge is how to optimally allocate edge cloud resources to competing user demands while considering both additive and sub-additive resources.

Abstracting the data/code for each task (or analytics) as a service and each attempt to access the data/code as a request, 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, 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 communication capacities, computation capacities, and other feasibility constraints (e.g., maximum tolerable latency), as well as a constraint that the scheduled edge cloud must have a replica of the requested service.

A. Related Work

Research on edge computing has evolved beyond point-to-point workload offloading. Instead of insisting each user to

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, either expressed or implied, 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.

always obtain service from its closest edge server, studies in [7], [8], [9] have shown that on the scale of metropolitan area networks (MAN), users can benefit from accessing services on edge servers that are not within their one-hop communication range. 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 multiple candidate edge servers per user opens up the problem of workload scheduling. There is a rich literature on edge workload scheduling, with various objectives (e.g., minimizing the cost [6] or the makespan [10]), workload models (e.g., fluid model [6], tasks [10], multi-component applications [11]), and edge cloud architectures (e.g., flat versus hierarchical [12]). These works all assume that different workloads are independent, each requiring its own resource for execution. While this assumption usually holds for computation and communication resources (assuming unicasts), it can be too restrictive for storage resources. In particular, in social applications such as social virtual reality and gaming, users on the same task often need to access the same data and code (e.g., scenes and object recognition for the scenes). Thus, one copy of the data/code can be used to serve multiple users, while individual user requests (e.g., scene update requests) still compete for CPU and bandwidth.

The problem of placing service data/code under storage constraints is similar to content placement in cache networks, especially in the case of proactive caching [13]. However, the content placement problem ignores the other resources (e.g., CPU, bandwidth) consumed in serving user requests from the cache. To our knowledge, no prior work has considered service provisioning in edge computing while considering both additive consumption of communication/computation resources and sub-additive consumption of storage resources. Although motivated by social applications, the fundamental problem we address applies to any applications with possibly shared data/code and non-shared computation and communication.

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 under limited communication, computation, and storage capacities per edge cloud.

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 (under a given service placement) can be computed in polynomial time by converting it to a maximum flow problem. We further show that 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 in comparison with benchmarks via trace-driven simulations. Our results show that the proposed solutions all achieve near-optimal performance in the number of served requests, while the greedy heuristic provides the best performance-complexity tradeoff.

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 with a discussion on future work. *All proofs are provided in [14].*

TABLE I
NOTATIONS

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

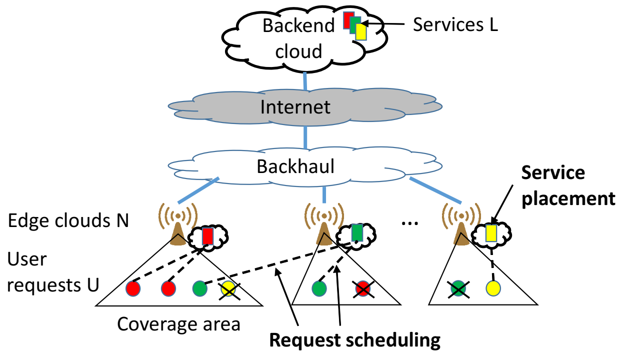


Fig. 1. System model, where different colors represent different types of services ($R = 1$, $W = 2$, $K = 3$).

II. Problem Formulation

We start by formally defining our model of the 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 an edge computing system in a local geographical region consisting of a set of edge clouds N, a set of services L (each encapsulates the data and code relevant to a task), and a set of user-generated service requests U (e.g., update the scene or recognize an object). 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 will be ignored) and the coverage areas are disjoint. For simplicity, we will refer to an edge cloud as a "cell" and a user request as a "user". The same user generating multiple requests will be modeled as multiple collocated users (possibly requesting different services).

Each cell contains certain communication, computation, and storage resources, and the cells in N form a resource pool that serves the users collaboratively. We assume that the cells are connected by backhaul links that can be used to send requests/responses between cells, which allows a user to be served by a non-local cell. Specifically, let $n_u \in N$ denote the cell covering user u, and $l_u \in L$ denote the service requested by user u. We allow the user to be served from any cell 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 any predetermined requirements on candidate servers, such as quality of service (QoS) 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 $U_n \triangleq \{u \in U : n_u = n\}$ denote the set of users located in cell n.

We consider three types of resource constraints: (1) *communication capacities* of the wireless access points limit the number of users each cell can admit (i.e., receive requests and return responses)[1], (2) *computation capacities* at the edge clouds limit the number of users each cell can serve (i.e., process requests), and (3) *storage capacities* at the edge clouds limit the number of service replicas each cell can store (and hence the number of different services a cell can provide). As a starting point, we consider the homogeneous case with identical capacities for all the cells 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 a cell, denoted by K; the computation capacity is the maximum number of users served by a cell, denoted by W; the storage capacity is the maximum number of different services a cell can provide, denoted by R. Note the distinction between "admit a user" and "serve a user". We will address the heterogeneous case in Section V.

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 cell, i.e., *service placement*, and how to match users with cells hosting their requested services, i.e., *request scheduling*. The latter also implies admission control, as users not scheduled to any cell will be dropped. To model these 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 cell n, and $y_{un} \in \{0, 1\}$ ($u \in U$, $n \in N$), which indicates whether user u is scheduled onto cell 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 decision 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 cell stores no more than R services and that a cell 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 cell, and constraint (1d) ensures that each cell admits no more than K users within its coverage (regardless of where they are scheduled). Finally, (1g) ensures that all the decision variables are indicators. While this formulation gives equal weights to all the users, it can be easily extended by adding weights to (1a) to differentiate the users.

Remark: Solutions to the static optimization (1) can be applied in an online setting by dividing the time into slots and applying the solutions to each slot based on predicted user demands. We have evaluated such scenarios in Section VI.B 3) in [14]. We leave a full-fledged study of the online setting to future work.

III. Complexity Analysis

The SPRS problem (1) is similar to knapsack problems where we want to pack the most items into a given set of knapsacks. However, while knapsack problems become easy when the items and the knapsacks are homogeneous, we will show that our problem is hard, even if some constraints are removed.

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]We focus on the last-hop communications between the users and their access points instead of backhaul communications, because the last-hop capacity (e.g., via IEEE 802.11b) is typically much smaller than the capacity in the backhaul (e.g., via wired links or IEEE 802.16).

1) Case 1: Removing K-constraints: If $K \geq |U|$ (i.e., no constraint on edge communications), then the ILP in (1) becomes:

$$\max \sum_{u\in U}\sum_{n\in N} y_{un} \tag{2a}$$
$$\text{s.t.} \sum_{n\in N} y_{un} \leq 1, \qquad \forall u \in U, \tag{2b}$$
$$\sum_{l\in L} x_{ln} \leq R, \qquad \forall n \in N, \tag{2c}$$
$$\sum_{u\in U} y_{un} \leq W, \qquad \forall n \in N, \tag{2d}$$
$$y_{un} \leq a_{un} \cdot x_{l_u n}, \qquad \forall u \in U,\ n \in N, \tag{2e}$$
$$x_{ln}, y_{un} \in \{0,1\}, \quad \forall l \in L, u \in U, , n \in N. \tag{2f}$$

We will show that the special case in (2) is at least as hard as the *3-partition problem* and thus NP-hard.

Theorem III.1. The ILP in (2) is NP-hard.

2) Case 2: Removing W-constraints: If $W \geq |U|$ (i.e., no constraint on computation), then the ILP in (1) becomes:

$$\max \sum_{u\in U}\sum_{n\in N} y_{un} \tag{3a}$$
$$\text{s.t.} \sum_{n\in N} y_{un} \leq 1, \qquad \forall u \in U, \tag{3b}$$
$$\sum_{l\in L} x_{ln} \leq R, \qquad \forall n \in N, \tag{3c}$$
$$\sum_{u\in U_n}\sum_{n'\in N} y_{un'} \leq K, \qquad \forall n \in N, \tag{3d}$$
$$y_{un} \leq a_{un} \cdot x_{l_u n}, \qquad \forall u \in U,\ n \in N, \tag{3e}$$
$$x_{ln}, y_{un} \in \{0,1\}, \quad \forall l \in L, u \in U, , n \in N. \tag{3f}$$

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 ILP in (3) is NP-hard.

3) Case 3: Removing R-constraints: If $R \geq |L|$ (i.e., every cell 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:

$$\max \sum_{u\in U}\sum_{n\in N} y_{un} \tag{4a}$$
$$\text{s.t.} \sum_{n\in N} y_{un} \leq 1, \qquad \forall u \in U, \tag{4b}$$
$$\sum_{u\in U_n}\sum_{n'\in N} y_{un'} \leq K, \qquad \forall n \in N, \tag{4c}$$
$$\sum_{u\in U} y_{un} \leq W, \qquad \forall n \in N, \tag{4d}$$
$$y_{un} \leq a_{un}, \qquad \forall u \in U, n \in N, \tag{4e}$$
$$y_{un} \in \{0,1\}, \qquad \forall u \in U, n \in N. \tag{4f}$$

In contrast to the previous cases, the special case in (4) 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).

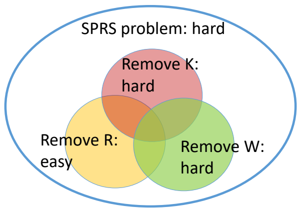


Fig. 2. Complexity of SPRS and its special cases.

Lemma III.3. The ILP in (4) is polynomial-time solvable.

B. General Case

The hardness of the special cases (2) and (3) 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 as shown in Section III motivates our search for efficient suboptimal solutions. To this end, we develop a polynomial-time solution with a constant-factor approximation to the optimum and an efficient heuristic.

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 cells. 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 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 located in cell 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., cell 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.

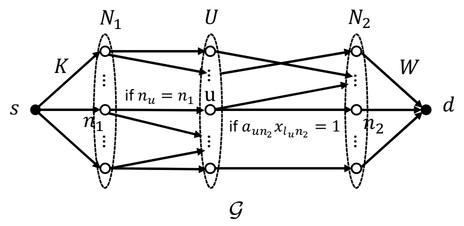


Fig. 3. Auxiliary graph $\mathcal{G}$ for a given service placement $\mathbf{x}$.

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;
```

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 [15] 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 $O(|N| \cdot |U|^2)$.

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 [16]). 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 [16]). Given a finite ground set E and a function $f : 2^E \to \mathbb{R}$,

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

If our objective function is monotone submodular and our constraint forms a matroid, then we can apply solutions from combinatorial optimization with a known approximation guarantee. In particular, 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 ([17]). 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 $f^g \ge f^*/2$.

We will show that 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 problem. Let $S(\mathbf{x}) \triangleq \{(l, n) \in L \times N : x_{ln} = 1\}$ denote the set of single-service placements selected according to $\mathbf{x}$, where $(l, n) \in S(\mathbf{x})$ means to place (a replica of) service l at cell n. Let $\Omega(S(\mathbf{x}))$ denote the optimal objective value of (1) for a given $\mathbf{x}$, i.e., the maximum number of users served under service placement $\mathbf{x}$. Simply writing $S(\mathbf{x})$ as S, we can rewrite the service placement subproblem as:

$$\max \Omega(S) \tag{5a}$$
$$\text{s.t. } |S \cap S_n| \le R, \qquad \forall n \in N, \tag{5b}$$
$$S \subseteq L \times N, \tag{5c}$$

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

- *Matroid constraint:* Let $\mathcal{I}$ be the collection of all the S's satisfying constraints (5b, 5c). 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 (5a) has the following properties.

Lemma IV.5. Function $\Omega(S)$ is monotone and submodular.

Algorithm: Since under the optimal request scheduling, the service placement subproblem is a maximization of a monotone submodular set function under a matroid constraint, we can apply existing approximation 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 until all the cells 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).

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

```
input : Input parameters of (1)
output: Service placement $\mathbf{x}$ and request scheduling $\mathbf{y}$
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 $(l^*, n^*) \neq \emptyset$ 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 $\mathbf{x}$;
```

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: Since our objective is monotone and submodular (Lemma IV.5), and our constraints form a matroid, we can apply Theorem IV.4 to guarantee the following.

Corollary IV.6. 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 number of users served under the optimal solution to SPRS (1).

Discussion: It is known that the optimal approximation ratio for maximizing a monotone submodular function under a matroid constraint is $1-1/e \approx 0.63$, achieved by a randomized algorithm given in [18]. In theory, we can apply this algorithm to service placement to achieve a better approximation ratio of $1-1/e$, using ORS (Algorithm 1) as a value oracle. In practice, however, applying this algorithm incurs a high complexity due to the large number of calls to Algorithm 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 cell n, $\widetilde{K}_n$ denoting the residual communication capacity at cell n, and $\widetilde{U}_{ln}$ denoting the set of unserved users in cell n requesting service l. Let $U_{ln} \triangleq \{u \in U : \; n_u = n, \; l_u = l\}$ denote all the users in cell n that request service l, and $V_n \triangleq \{u \in U : \; a_{un} = 1\}$ denote all the users that can be served by cell 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 $\widetilde{U}_{ln}$.

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;
```

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

$$\min(\widetilde{W}_n, \sum_{n' \in N} \min(|\widetilde{U}_{ln'} \cap V_n|, \; \widetilde{K}_{n'})) \tag{6}$$

new users without rescheduling the existing users. Let

$$\Phi(\mathbf{x}) \triangleq \{(l, n) \in L \times N : \; x_{ln} = 0, \; \sum_{l' \in L} x_{l'n} < R\} \tag{7}$$

denote the set of feasible placements of one additional service. GSP-GRS selects the placement in $\Phi(\mathbf{x})$ that maximizes (6) (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 cells 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 cell $n \in N$, we sort the services into descending order of the fractional x_{ln}'s ($\forall l \in L$), and place the top R services in cell n; for each user $u \in U$, we sort the cells into descending order of the fractional y_{un}'s ($\forall n \in N$), and schedule u to the first available cell (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* [19]. 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 cells 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 cells, 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 cell n, which can be different for different cells. 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*:

$$\max \sum_{u \in U} \sum_{n \in N} y_{un} \tag{8a}$$

$$\text{s.t.} \sum_{n \in N} y_{un} \le 1, \qquad \forall u \in U, \tag{8b}$$

$$\sum_{l \in L} x_{ln} \cdot r_l \le R_n, \qquad \forall n \in N, \tag{8c}$$

$$\sum_{u \in U_n} \Big(\sum_{n' \in N} y_{un'}\Big) \cdot \kappa_{l_u} \le K_n, \qquad \forall n \in N, \tag{8d}$$

$$\sum_{u \in U} y_{un} \cdot \omega_{l_u} \le W_n, \qquad \forall n \in N, \tag{8e}$$

$$y_{un} \le a_{un} \cdot x_{l_u n}, \qquad \forall u \in U,\ n \in N, \tag{8f}$$

$$x_{ln}, y_{un} \in \{0,1\}, \qquad \forall l \in L, u \in U, n \in N. \tag{8g}$$

B. Complexity Analysis

As (8) 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 (8) is still NP-hard.*

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 [14] 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 [14] for additional results from synthetic simulations.

A. Benchmarks

As benchmarks for our algorithms, we evaluate: (1) the optimal solution obtained by solving (1, 8) using an ILP solver, and (2) a baseline solution that first places a randomly selected subset of services in each cell, and then schedules as many users as possible under the given service placement (via ORS in the homogeneous case and greedy scheduling in the heterogeneous case). The latter is used to evaluate the performance gain via jointly optimizing service placement and request scheduling, as opposed to only optimizing request scheduling.

B. Simulation Setting

We extract user and cell locations from real mobility traces and cell tower locations. We use the taxi cab traces from [20], from which we extract the traces of 280 users (i.e., taxi cabs) over a 100-minute period with location updates every minute[2]. We quantize the user locations into Voronoi cells based on cell tower locations 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 only use the occupied cells to offer services (plus some unoccupied cells if there are fewer than 6 occupied cells in a slot)[3]. In each slot, we synthetically generate a request for each user 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.

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

[3]There are many more (> 280) cell towers in the area covered by the traces, and thus the resource provisioning problem will become trivial if we use all the cells to offer services.

TABLE II
RUNNING TIME FOR PROPOSED ALGORITHMS VS. OPTIMAL SOLUTION

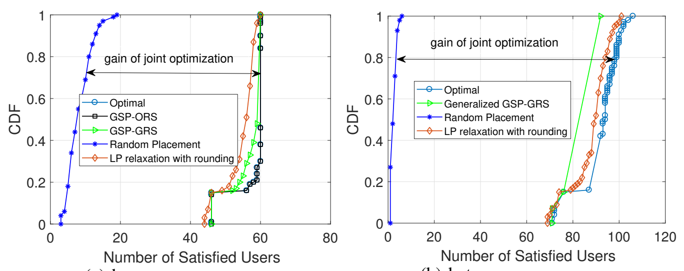
Algorithm	Average Running time
Optimal via brute-force search[4]	8.7357e+75 sec
GSP-ORS	535.9117 sec
GSP-GRS	0.0188 sec
LP relaxation with rounding	0.9852 sec
Random placement with best-effort scheduling	0.0277 sec

C. Evaluation Results

1) Performance Comparison: Fig. 4 shows the cumulative distribution function (CDF) of the number of served users over the 100 slots. In the homogeneous case (Fig. 4(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 8.67 for random placement; in the heterogeneous case (Fig. 4(b)), this number is 92.22 for the optimal solution, 89.25 for (the generalized) GSP-GRS, 88.19 for LP relaxation, and 2.63 for random placement.

In both cases, there is a huge gap between the random placement-based solution that only optimizes request scheduling and the other solutions that jointly consider service placement and request scheduling, which signals the importance of a proper service placement. Moreover, 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. 4(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 ($\sim$ 50X faster than LP relaxation). We note that the presented running times are based on our initial implementation in Matlab, and can be improved via more efficient implementation methods.



(a) homogeneous case (b) heterogeneous case

Fig. 4. Performance comparison in trace-driven simulations.

VII. CONCLUSION

We have studied the problem of joint service placement and request scheduling in edge computing systems under communication, computation, and storage constraints. Through a comprehensive 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 achieve near-optimal performance while significantly reducing the running time compared to the brute-force-based optimal solution. We conclude by noting that while the proposed solutions can be applied in an online setting based on predicted user demands, the online setting introduces new considerations such as the adaptation cost and the uncertainty in user demands, which we leave to future work.

[4]This is estimated by multiplying the number of all possible service placements with the time to compute the optimal request scheduling, estimated by the running time of ORS in the homogeneous case (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.

REFERENCES

[1] 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.

[2] "Open edge computing." [Online]. Available: openedgecomputing.org/

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

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

[13] 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.

[14] "It's Hard to be Social: Joint Service Placement and Request Scheduling for Social Edge Applications," Technical Report, July 2017. [Online]. Available: https://1drv.ms/b/s!Av2FAzDDqJorapcryHJRTZj7C5k

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

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

[17] 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.

[18] 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.

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

[20] 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