Service Placement and Request Scheduling for Data-intensive Applications in Edge Clouds

Vajiheh Farhadi, Fidan Mehmeti, Ting He, *Senior Member, IEEE,* Tom La Porta, *Fellow, IEEE*, Hana Khamfroush, Shiqiang Wang, Kevin S Chan, *Senior Member, IEEE*, and Konstantinos Poularakis

***Abstract*—Mobile edge computing provides the opportunity for wireless users to exploit the power of cloud computing without a large communication delay. To serve data-intensive applications (e.g., video analytics, machine learning tasks) from the edge, we need, in addition to computation resources, storage resources for storing server code and data as well as network bandwidth for receiving user-provided data. Moreover, due to time-varying demands, the code and data placement needs to be adjusted over time, which raises concerns of system stability and operation cost. In this paper, we address these issues by proposing a two-time-scale framework that jointly optimizes service (code and data) placement and request scheduling, while considering storage, communication, computation, and budget constraints. First, by analyzing the hardness of various cases, we completely characterize the complexity of our problem. Next, we develop a polynomial-time service placement algorithm by formulating our problem as a set function optimization, which attains a constant-factor approximation under certain conditions. Furthermore, we develop a polynomial-time request scheduling algorithm by computing the maximum flow in a carefully constructed auxiliary graph, which satisfies hard resource constraints and is provably optimal in the special case where requests have homogeneous resource demands. Extensive synthetic and trace-driven simulations show that the proposed algorithms achieve 90% of the optimal performance.**

***Index Terms*—Mobile edge computing, service placement, workload scheduling, complexity analysis, algorithm design.**

I. Introduction

The emerging technology of *mobile edge computing* [3] enables wireless users to run resource-intensive and delay-sensitive applications from the edge of mobile networks, at small server clusters referred to as *edge clouds* [4], *cloudlets* [5], *fog* [6], *follow me cloud* [7], or *micro clouds* [8]. Mobile applications are increasingly resource-demanding as they address use cases based on big data and machine learning problems. As users access these resource-hungry applications via bandwidth-limited wireless links, how to optimally allocate the limited resources at edge clouds to competing demands poses a difficult but intriguing research question, which has attracted tremendous interest in the research community.

Intuitively, one should strive to serve every user request from the nearest edge cloud. While this intuition has been supported by empirical studies [9], maintaining service locality for mobile users poses a significant challenge, including how to migrate services [10] and when/where to migrate services [11], [12], [4], [13], in order to attain a desirable tradeoff between the quality of service and the migration cost. When some of the edge clouds are heavily loaded, it has been shown that users can benefit from getting served by non-nearest edge clouds in the same metropolitan area network [14], [15], [16]. Meanwhile, there have been standardization initiatives [17], [18], [19] to create an open edge computing environment, such that edge clouds within the same geographical region form a shared resource pool, which can then be distributed among contending user requests. The existence of a shared resource pool creates the need for *request scheduling*, i.e., on which edge server, if any, to schedule each user request such that a given objective can be optimized [20], [21]. Existing works typically assume that serving each request needs a *dedicated* share of resources including CPU cycles, memory space, and network bandwidth, and that the total resource consumption at a server is the summation of resource demands scheduled to it.

While the above assumption holds for applications that do not need notable amounts of data on the server, it fails to capture the requirements of *data-intensive applications*. In such applications (e.g., video analytics, distributed machine learning), serving a request needs both a dedicated amount of resources and a significant amount of data at the server (e.g., object database, trained machine learning models). The storage resource for storing such data fundamentally differs from the other types of resources in that it is *amortized* over all requests against the same copy of data. Note that many data-intensive applications also require a nontrivial amount of user-provided data (e.g., images captured by the user), although the resources for collecting/storing such data are typically dedicated to each request. Hence, in addition to the conventional resources of CPU cycles and memory, resource allocation algorithms for data-intensive applications should also consider the storage resources for storing server data and the network bandwidth for receiving user-provided data.

Jointly allocating dedicated and amortized resources induces a decomposition of the resource allocation problem into two subproblems [2]: (i) *service placement*, which decides how to replicate and place each service (including server code and data) within the storage capacity of each edge cloud, and (ii) *request scheduling*, which decides whether/where to schedule each request within the communication and the computation capacities of edge clouds, as well as other constraints (e.g.,

V. Farhadi, F. Mehmeti, T. He, and T. La Porta are with Pennsylvania State University, University Park, PA 16802 USA (email: {vuf8, fzm82, tzh58, tlp}@cse.psu.edu).
H. Khamfroush is with University of Kentucky, Lexington, KY 40506 USA (email: khamfroush@cs.uky.edu).
S. Wang is with IBM T. J. Watson Center, Yorktown Heights, NY 10598 USA (email: wangshiq@us.ibm.com).
K. Chan is with Army Research Laboratory, Adelphi, MD 20783 USA (email: kevin.s.chan.civ@mail.mil).
K. Poularakis is with Yale University, New Haven, CT 06520 USA (email: kpoularakis@gmail.com).

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.

Aspects of this work have been published at IEEE INFOCOM'19 [1] and IEEE ICDCS'18 [2].

maximum delay). The two subproblems are coupled by the fact that the edge cloud scheduled to process a request must have a replica of the requested service. The existing solution [2] makes both decisions at the same time, and thus may adjust service placement as frequently as the scheduling of requests, risking a high operation cost and even system instability.

In this work, we jointly consider service placement and request scheduling for data-intensive applications. In contrast to [2], we separate the time scales of the two decisions: to prevent system instability, service placement happens at a larger scale (*frames*); to limit scheduling delay, request scheduling happens at a smaller scale (*slots*). Frame length is tuned based on dynamics of user mobility and user request patterns, while slot length is tuned based on the desired scheduling delay and expected execution time of jobs. Furthermore, to control the operation cost due to service replication/migration, we impose a budget constraint for service placement. For the request scheduling subproblem, we separately consider a version with *soft constraints* where the resource capacities are only enforced on the average, and a version with *hard constraints* where the resource capacities are strictly enforced.

A. Related Work

While early works on mobile edge computing assumed that every user can only access its closest edge server, studies in [14], [15], [16] have shown that users can benefit from accessing services on edge servers that are multiple hops away. Allowing the use of non-local edge servers created the problem of edge workload scheduling, which has attracted significant attention in recent years. Existing works have used various objectives (e.g., minimizing the cost [20] or the makespan [21]), workload models (e.g., fluid model [20], tasks [21], multi-component applications [22]), and edge cloud architectures (e.g., flat versus hierarchical [23]). These works typically assume that each workload requires its own resource for execution, i.e., the resources are dedicated. Here "dedicated" means that each unit of resource can only be used by one workload, e.g., two workloads can share a processor but each CPU cycle is only used by one workload. While this assumption usually holds for computation and communication resources (assuming unicast), it can be too restrictive for storage resources. Note that although [21] allows each service replica to serve multiple jobs, it does not optimize the service placement.

Meanwhile, works on content placement in cache networks have considered storage resources that can be shared among requests for the same content. Various solutions have been developed to place contents under cache capacity constraints based on predicted content popularities [24], [25] or request history [26]. Variations of the problem have been studied, e.g., a cache can serve requests from other caches [27], or the content placement and the routing of requests can be jointly optimized [28]. However, the content placement problem only considers the storage resource (i.e., cache space), while the other types of resources (e.g., CPU, bandwidth) are ignored. Note that although [26] was motivated by "hosting services", the problem was actually about caching. In the spirit of content placement, [29] studied the placement of service replicas to minimize the latency. However, request scheduling was not considered in this work.

Only a few works have considered multiple types of resources (e.g., storage, computation, communication). In [30], [31], mixed integer linear programs (MILPs) were formulated for placing content or service functions, and activating storage, computation, and communication resources in a distributed cloud network. However, no formal complexity analysis or algorithm with performance guarantees was provided. In [32], a dynamic service placement and workload scheduling framework was proposed to jointly allocate storage and computation resources, but it assumes that requests can be queued indefinitely and does not consider bandwidth constraints. In [33], an algorithm with performance guarantee was developed for placing virtual network functions (VNFs) in distributed cloud networks and routing service flows among the placed VNFs under chaining constraints. However, each unit of resource (CPU, memory, bandwidth) is dedicated to a flow (i.e., not amortized), and there is no storage capacity constraint on the VNF placement. In [34], an optimal algorithm was developed for joint resource placement and assignment in distributed networks, where a "resource" means a service, and a "type of resources" means a type of services. The solution actually assumed that each placed service can only serve one request (i.e., dedicated). In [35], the authors consider the problem of resource provisioning and replica placement in cloud CDNs, where the objective is to minimize the cost, whereas our objective is to maximize the expected number of served requests per time slot. Furthermore, all the requests in [35] are of equal size and equal computation requirement, as opposed to our setup where these parameters can be different for different requests.

The work closest to ours is [36], which considered joint service placement and request routing in multi-cell edge cloud systems. However, it assumed that all the resources consumed by a request (bandwidth, computation, storage) are associated with a single edge cloud, as opposed to our setup where the bandwidth consumption is with the edge cloud directly covering the user, and the other resources are with the (possibly different) edge cloud selected to process the request. As explained in Section III, this leads to critical differences in the complexity of the underlying optimization problem.

B. Summary of Contributions

The main contributions of this paper are as follows:

1) We propose a two-time-scale framework for joint service placement and request scheduling, and formulate the underlying optimization as a mixed integer linear program (MILP) that jointly considers dedicated and amortized resources.

2) By examining the complexity of our problem in carefully selected special cases, we not only prove that both the service placement subproblem and the request scheduling subproblem (under hard constraints) are generally NP-hard, but also determine all the cases that are polynomial-time solvable and identify the root cause of the hardness.

3) By reformulating the service placement subproblem as a set function optimization, we develop a greedy service placement algorithm based on shadow request scheduling computed by a linear program (LP). By proving that our objective function is monotone sub-modular under certain conditions and our constraints form a p-independence system, we derive a constant-factor approximation guarantee for the proposed algorithm.

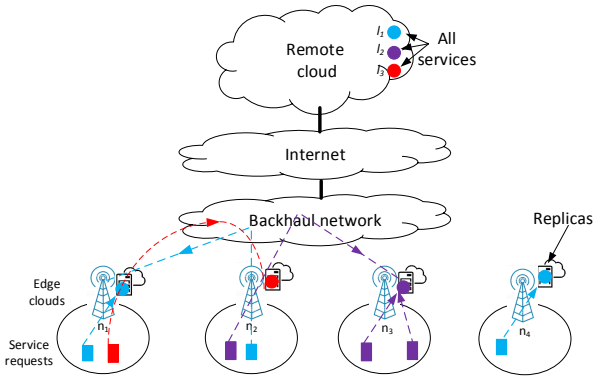


Figure 1. System model.

4) We show that in the special case where all the requests demand the same amount of communication/computation resources, the request scheduling subproblem under hard constraints can be converted to a maximum flow problem in a carefully constructed auxiliary graph, based on which we develop a polynomial-time algorithm that is provably optimal.

5) We show that both our formulation and our service placement algorithm can be extended to exploit request prediction over multiple frames.

6) We perform extensive performance evaluations via synthetic and trace-driven simulations. The evaluations show that: (i) the key performance differentiator is the service placement algorithm (i.e., a simplistic algorithm suffices for request scheduling), (ii) the proposed service placement algorithm consistently outperforms benchmarks and achieves over 90% of the optimal performance, even when the approximation guarantee does not hold, and (iii) the performance can be notably improved by jointly planning service placements for multiple frames, while most of the improvement is already achieved by considering two frames at a time.

Roadmap. Section II formulates our problem within a single frame, for which Section III analyzes the complexity, and Section IV presents our algorithms and their performance analysis. Section V extends our solution to multiple frames. Section VI evaluates the proposed solution against benchmarks. Finally, Section VII concludes the paper.

II. Problem Formulation

A. System Model

As illustrated in Fig. 1, we consider a wireless edge network consisting of a set N of edge clouds, each accessible via a wireless access point or base station covering a specified area. We assume that all the edge clouds are connected by back-haul links that can be used for inter-cloud communications. There is a set L of services, of which a subset can be hosted by each edge cloud at a given point in time, subject to storage capacity constraints.

Services may be migrated/replicated between edge clouds, and/or from a remote cloud to an edge cloud. To access a certain service, a user will first send a request for this service to its local edge cloud, which may then route the request to another edge cloud for processing. Serving a request for service l submitted to edge cloud n at edge cloud m (possibly $m \neq n$) consumes communication resources for transferring input/output between the user and edge cloud n, and computation resources at edge cloud m. Additionally,

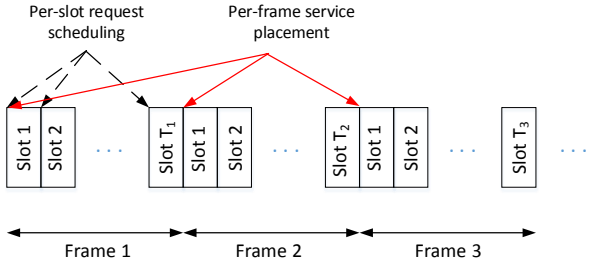


Figure 2. Time scales of service placement and request scheduling.

edge cloud m must have a replica of service l. If $m \neq n$, communication resources are also consumed for transferring input/output between edge cloud n and edge cloud m, but as back-haul links usually have much higher bandwidth than access links, we will focus on the communication resources consumed at the access link in edge cloud n.

To ensure system stability while providing timely services, we adopt a two-time-scale framework as illustrated in Fig. 2, where service placement is performed once per frame at the beginning of the frame, and request scheduling is performed once per slot at the beginning of the slot. We discuss how to tune the values of frame and slot length in Section VI. Furthermore, we impose a budget B to control the cost of migrating/replicating services in each frame. We refer to a request for service l that is submitted to edge cloud n as a "type-(l,n)" request. The average rate of type-(l,n) requests in frame f is denoted by λ_{ln}^{f} (unit: requests/slot), which is assumed to be predictable based on the request history [37], [38], [39]. The actual number of type-(l,n) requests in slot t is denoted by λ_{ln}^{t}, which is only known at the beginning of slot[1] t. We evaluate the impact of predicting the request rate in Section VI.

Each edge cloud has limited communication, computation, and storage capacities. The capacities of different edge clouds may be different. Likewise the size of each service replica and the communication/computation resources required by each request may be different. There may be other constraints (e.g., latency, security) on whether a given edge cloud m is permitted to serve type-(l,n) requests, and we model that by an indicator a_{lnm} ('0': not permitted; '1': permitted). The main notations used in this paper are described in Table I.

In the following subsection, we provide the mathematical formulation of the optimization problems.

B. Underlying Optimization Problems

We now formulate the optimization problems for service placement and request scheduling formally. The service placement problem is solved once per frame; once the placement is set, the request scheduling problem is solved once per slot within the frame. Variables related to the frame are denoted with f and those related to a slot are denoted with t.

For the scheduling problem we consider both soft and hard constraints. For soft constraints we use probabilistic scheduling knowing that in some cases requests will not be accommodated within their slot but must be served in a subsequent slot. For hard constraints we guarantee that all scheduled requests are served within the next slot.

[1]This is feasible by considering all the requests received during slot $t-1$ as being "submitted" in slot t.

Table I
TABLE OF NOTATIONS

Notation	meaning
N	set of edge clouds
$N_+ = N \cup \{n_0\}$	set of edge clouds plus the remote cloud n_0
L	set of all possible services
R_n	storage capacity of edge cloud n
W_n	processing capacity of edge cloud n (per slot)
K_n	communication capacity of edge cloud n (per slot)
r_l	size per replica of service l
κ_l	size of input/output data per request for service l
ω_l	computation requirement per request for service l
$a_{lnm} \in \{0,1\}$	indicates whether edge cloud m is permitted to serve type-(l,n) requests
$\lambda^t_{ln}, \lambda^f_{ln}$	actual number of type-(l,n) requests in slot t and average number of type-(l,n) requests per slot in frame f
$c_{ln'n}$	cost of replicating or migrating service l from cloud n' to edge cloud n, where cloud n' can be either a remote cloud or an edge cloud
B	maximum cost for service placement in one frame
$x^f_{ln} \in \{0,1\}$	placement variable for frame f, 1 if service l is placed on edge cloud n and 0 otherwise
$y^t_{lnm}, y^f_{lnm} \in [0,1]$	scheduling variable representing the probability that a type-(l,n) request is scheduled to edge cloud m in slot t (under soft resource constraints) or frame f
z^t_{lnm}	scheduling variable representing the number of type-(l,n) requests that are scheduled to edge cloud m in slot t (under hard resource constraints)

1) Service Placement with Shadow Scheduling: To evaluate the service placement cost, we assume that the services always exist on the remote cloud n_0, i.e., $x^f_{ln_0} \equiv 1$, and deleting a service replica from an edge cloud incurs no cost. Furthermore, we always replicate a service from the nearest location hosting the service. That is, the cost of placing service l at edge cloud n in frame f is $c^f_{ln} = \min_{n' \in N_+ : x^{f-1}_{ln'}=1} c_{ln'n}$, where $c_{lnn} \equiv 0$.

The optimization problem for service placement can be formulated as (1): Objective (1a) maximizes the expected number of requests served per slot. Constraint (1b) guarantees that the scheduling variables are valid. Constraint (1c) ensures that each edge cloud n does not store more than its storage capacity R_n. Constraint (1d) guarantees that the total communication demand on an edge cloud n stays within its communication capacity K_n on the average. Constraint (1e) ensures that the total computation demand scheduled to an edge cloud m is within its computation capacity W_m on the average. Constraint (1f) states that an edge cloud can only serve a request if it contains the requested service and is a candidate server. Constraint (1g) ensures that the total service placement cost is within the budget. Constraint (1h) specifies valid ranges of the decision variables. In the sequel, and especially in Section III, we will refer to constraint (1c) as *R-constraint*, (1d) as *K-constraint*, (1e) as *W-constraint*, and (1g) as *B-constraint*.

At the beginning of each frame f, we solve (1) with the predicted request rates[2] $\lambda_{ln} = \lambda^f_{ln}$ and the placement costs $c_{ln} = c^f_{ln}$ for the service placement x^f_{ln} and the corresponding request scheduling y^f_{lnm}. Then only x^f_{ln} will be used (to place services). Although the scheduling variable y^f_{lnm} will not be used for actual scheduling, it is needed to evaluate the served request rate (1a) under a given service placement. For this reason, we refer to y^f_{lnm} as the *shadow scheduling variable*.

[2]Recall that the superscript f indicates parameters/variables corresponding to frame f, whereas the superscript t parameters corresponding to slot t.

$$\max \sum_{l\in L}\sum_{n\in N}\lambda_{ln}\sum_{m\in N} y_{lnm} \tag{1a}$$

$$\text{s.t.} \sum_{m\in N} y_{lnm} \le 1, \qquad \forall l \in L, n \in N, \tag{1b}$$

$$\sum_{l\in L} x_{lm} r_l \le R_m, \qquad \forall m \in N, \tag{1c}$$

$$\sum_{l\in L}\lambda_{ln}\kappa_l \sum_{m\in N} y_{lnm} \le K_n, \qquad \forall n \in N, \tag{1d}$$

$$\sum_{l\in L}\omega_l \sum_{n\in N}\lambda_{ln} y_{lnm} \le W_m, \qquad \forall m \in N, \tag{1e}$$

$$y_{lnm} \le a_{lnm} x_{lm}, \qquad \forall l \in L, n \in N, m \in N, \tag{1f}$$

$$\sum_{l\in L}\sum_{n\in N} x_{ln} c_{ln} \le B, \tag{1g}$$

$$x_{ln} \in \{0,1\}, y_{lnm} \ge 0, \qquad \forall l \in L, n \in N, m \in N. \tag{1h}$$

2) Request Scheduling under Soft Resource Constraints: Depending on whether requests submitted in a slot can be postponed till a later slot, the optimization problem for request scheduling differs slightly. If the requests can be postponed, then the average resource constraints (1d,1e) suffice, as temporary bursts in demands can be absorbed over time. In this case, at the beginning of each slot t within frame f, we solve (1) with the current demands $\lambda_{ln} = \lambda^t_{ln}$ and the previously determined service placement $x_{lm} = x^f_{lm}$ for the scheduling variable y^t_{lnm}, which is then used to schedule requests probabilistically in this slot.

3) Request Scheduling under Hard Resource Constraints: If the requests cannot be postponed, e.g., for services with hard deadlines, then we must impose hard resource constraints such that all the requests scheduled to the edge clouds in slot t can be finished within the same slot (the unscheduled requests will be routed to the remote cloud for processing). The corresponding optimization problem can be formulated as (2) ($\mathbb{N}$: natural numbers):

$$\max \sum_{l\in L}\sum_{n\in N}\sum_{m\in N} z_{lnm} \tag{2a}$$

$$\text{s.t.} \sum_{m\in N} z_{lnm} \le \lambda_{ln}, \qquad \forall l \in L, n \in N, \tag{2b}$$

$$\sum_{l\in L}\kappa_l \sum_{m\in N} z_{lnm} \le K_n, \qquad \forall n \in N, \tag{2c}$$

$$\sum_{l\in L}\omega_l \sum_{n\in N} z_{lnm} \le W_m, \qquad \forall m \in N, \tag{2d}$$

$$z_{lnm} \le a_{lnm} x_{lm} \lambda_{ln}, \qquad \forall l \in L, n \in N, m \in N, \tag{2e}$$

$$z_{lnm} \in \mathbb{N}, \qquad \forall l \in L, n \in N, m \in N. \tag{2f}$$

Optimization (2) is similar to (1) under a fixed feasible service placement x_{lm}, after replacing $\lambda_{ln} y_{lnm}$ by a new variable z_{lnm}. The only difference is that we now impose an integer constraint (2f), which means that instead of only specifying the expected number of type-(l,n) requests to schedule to edge cloud m (i.e., $\lambda_{ln} y_{lnm}$), we specify the exact number (i.e., z_{lnm}). At the beginning of each slot t, we solve (2) with the demand $\lambda_{ln} = \lambda^t_{ln}$ and the service placement $x_{lm} = x^f_{lm}$ (f: the current frame) for the scheduling variable z^t_{lnm}, which is used to schedule requests deterministically in this slot. The deterministic scheduling ensures that instead of satisfying the communication/computation capacities on the average as in (1d,1e), we now satisfy them strictly, which ensures that all the scheduled requests can finish within the slot. Note also that since we solve optimization problem (2)

under a given feasible service placement $x_{l,m}$, constraints (1c) and (1g) are lifted as the service placement already satisfies those two constraints.

Discussion: At each decision epoch of service placement, we only know the average request rates over the next frame, and thus cannot impose the hard resource constraints. Therefore, soft constraints are assumed for the shadow scheduling problem to evaluate the objective (1a) under a given service placement.

III. Complexity Analysis

The service placement problem (1) is a *mixed integer linear program (MILP)*, and the request scheduling problem is a *linear program (LP)* under soft constraints and an *integer linear program (ILP)* under hard constraints. While LP can always be solved in polynomial time [40], MILP and ILP can both be NP-hard [41]. We thus need to understand the complexity of our instances of the MILP/ILP problem.

A. Complexity of Service Placement

The service placement problem (1) is related to, but different from several known problems in the literature, including the knapsack problem, the *data placement problem (DPP)* [42], the *generalized assignment problem (GAP)* [43], the *distributed caching problem (DCP)* [44]. These problems can all be seen as special cases of the *separable assignment problem (SAP)* [44]. SAP considers packing items into bins under general packing constraints that can model both dedicated and non-dedicated resources. For example, if items represent requests and bins represent edge clouds, then SAP can model service placement where requests for the same service can share a service replica, while each consuming a dedicated share of computation resource and bandwidth. However, SAP requires all the resources consumed for serving a request to be with a single edge cloud as the packing constraints for different bins (each representing an edge cloud) are imposed independently. This requirement is satisfied by [36], but not our problem.

We analyze the complexity of (1) by considering important special cases. In this optimization problem, there are four types of resource constraints: the R-constraint (1c), the K-constraint (1d), the W-constraint (1e), and the B-constraint (1g).

1) Having B-constraint Only: Consider the special case where the edge clouds and the services are homogeneous (although having B-constraint only gives the same formulation for homogeneous and heterogeneous scenarios), and R, W and K are large enough that they are unconstrained, i.e., $R \geq |L|$ (i.e., every edge cloud can store all the services), $W \geq \sum_{n\in N}\sum_{l\in L}\lambda_{ln}$ and $K \geq \max_{n\in N}\sum_{l\in L}\lambda_{ln}$. Then, the MILP in (1) changes to:

$$\max \sum_{l\in L}\sum_{n\in N}\lambda_{ln}\sum_{m\in N} y_{lnm} \tag{3a}$$

$$\text{s.t. } (1b),(1f),(1g), \tag{3b}$$

$$x_{ln}\in\{0,1\}, y_{lnm}\in[0,1], \qquad \forall l\in L, n\in N, m\in N. \tag{3c}$$

Theorem 1. *The B-constraint alone makes the problem NP-hard.*

Proof. We prove the NP-hardness of (3) by a reduction from the 0-1 knapsack problem: given a set of k items, each with value v_i and weight w_i $(i = 1, ..., k)$, select a subset $\mathcal{S}'$ such that $\sum_{i\in\mathcal{S}'} v_i$ is maximized while $\sum_{i\in\mathcal{S}'} w_i \leq \Omega$, for a given size Ω of the knapsack. The problem is well-known to be NP-hard [45].

Construction: For each item i, construct a service l_i with total demands $\sum_{n\in N}\lambda_{l_i n} = v_i$ and the placement cost $c_{l_i n} = w_i, \forall n \in N$. Let $B = \Omega$ and $a_{lmn} \equiv 1$.

Claim: The optimal service placement of (3) gives the optimal solution to a knapsack problem.

Proof of the claim: The optimal service placement places at most one replica among all the edge clouds. Therefore, the scheduling decision is to simply schedule all the requests for service l_i to edge cloud n, if $\exists n \in N$ with $x_{l_i n} = 1$; or, not schedule any of these requests if $x_{l_i n} = 0, \forall n \in N$. Let $\mathcal{S}'$ be the set of indices of all the placed services under the optimal solution to (3). Then, the expected number of served requests equals $\sum_{i\in\mathcal{S}'} v_i$, and $\sum_{i\in\mathcal{S}'} w_i \leq B = \Omega$. Selecting all the items corresponding to the services placed by the optimal solution of (3) provides the optimal solution to the knapsack problem. □

Remark: Proving NP-hardness for the special case shows that the problem is NP-hard in the general case as well.

2) Having R-constraint Only: Here we consider the special case in which the edge clouds and the services are homogeneous, and W, K and B are large enough to be unconstrained, i.e., $W \geq \sum_{n\in N}\sum_{l\in L}\lambda_{ln}$, $K \geq \max_{n\in N}\sum_{l\in L}\lambda_{ln}$, and $B \geq \sum_{l\in L}\sum_{n\in N} c_{ln}$. In this case, the MILP in (1) becomes:

$$\max \sum_{l\in L}\sum_{n\in N}\lambda_{ln}\sum_{m\in N} y_{lnm} \tag{4a}$$

$$\text{s.t. } (1b),(1f),(3c), \tag{4b}$$

$$\sum_{l\in L} x_{ln} \leq R, \qquad \forall n \in N. \tag{4c}$$

Theorem 2. *The R-constraint alone makes the problem NP-hard.*

Proof. We prove the hardness by showing that the optimization (4) can be reduced to the 2-Disjoint Set Cover ($2DSC$) problem, which is proved to be NP-complete [46]. Given a bipartite graph $\mathcal{G} = (\mathcal{A}, \mathcal{B}, \mathcal{E})$, with edges $\mathcal{E}$ between two disjoint vertex sets $\mathcal{A}$ and $\mathcal{B}$, $2DSC$ determines whether there exist two disjoint sets $\mathcal{B}_1, \mathcal{B}_2 \subset \mathcal{B}$, such that $|\mathcal{B}_1| + |\mathcal{B}_2| = |\mathcal{B}|$ and $\mathcal{A} = \cup_{b\in\mathcal{B}_1}\mathcal{N}(b) = \cup_{b\in\mathcal{B}_2}\mathcal{N}(b)$, where $\mathcal{N}(b)$ $(\forall b \in \mathcal{B})$ is the set of neighbors of node b.

Construction: Denote $\mathcal{A}$ by $\{a_1, ..., a_I\}$ and $\mathcal{B}$ by $\{b_1, ..., b_J\}$. WLOG, assume $I \leq J$. Construct J edge clouds $N = \{n_1, ..., n_J\}$, each with $R = 1$. Construct two services $L = \{l_1, l_2\}$, each with a unit of demand in the first I edge clouds, i.e., $\lambda_{ln_i} = 1, \forall i \in \{i, ..., I\}, l \in \{l_1, l_2\}$. Note that $\lambda_{ln_i} = 0, \forall i > I$. For each $i \in \{1, ..., I\}$ and $j \in \{1, ..., J\}$, we allow edge cloud n_j to serve requests of type (l_1, n_i) and (l_2, n_i), if and only if $(a_i, b_j) \in \mathcal{E}$, i.e., $a_{l_k n_i n_j} = 1$, $k = \{1, 2\}$, if $(a_i, b_j) \in \mathcal{E}$, otherwise $a_{l_k n_i n_j}$ is zero.

Claim: $2DSC$ is feasible if and only if the optimal value of (4) for the above instance is $2\mathcal{I}$.

Proof of the claim: If $2DSC$ is feasible, then storing l_1 at edge clouds corresponding to $\mathcal{B}_1$ and l_2 at the remaining edge clouds will serve all the requests. If there is a service placement that serves all the requests, then $\mathcal{B}_1 = \{b_i \in \mathcal{B} : n_i \text{ stores } l_1\}$, and $\mathcal{B}_2 = \mathcal{B}\backslash\mathcal{B}_2$ is a feasible solution to $2DSC$. □

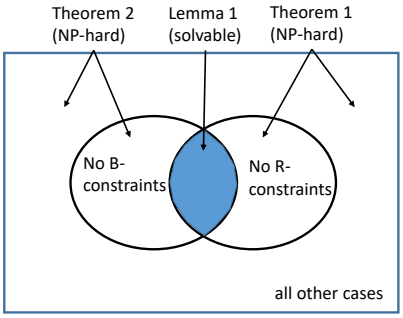


Figure 3. Complexity of service placement (1).

3) Removing R- and B-constraints:

Lemma 1. *Removing R- and B-constraints makes the problem polynomial-time solvable.*

Proof. If R_n ($\forall n \in N$) and B are both large enough, i.e., $\min_{n\in N} R_n \geq |L|$ (every edge cloud can store all the services) and $B \geq \sum_{l\in L}\sum_{n\in N} c_{ln}$, the optimal solution to x_{ln} is trivially $x_{ln} \equiv 1$ ($\forall l \in L$ and $n \in N$). Under this service placement, constraints (1c,1g) in (1) disappear, and constraint (1f) changes to $y_{lnm} \leq a_{lnm}$ ($\forall l \in L, n \in N, m \in N$). Removing the constraints (1c,1g) reduces the original problem (1) into a linear program (LP), which is polynomial-time solvable [40]. □

4) Summary of All Cases: Together, Theorems 1, 2 and Lemma 1 cover all the cases. By Theorem 1, the solvable instances must be cases without the B-constraint. By Theorem 2, the solvable instances must also be cases without the R-constraint. On the other hand, Lemma 1 shows that all the cases without either of B- or R-constraint are polynomial-time solvable. Therefore, the colored region in Fig. 3 captures all the solvable cases of (1).

B. Complexity of Request Scheduling

Under soft resource constraints, the request scheduling problem ((1) with given x_{lm}) is an LP and hence polynomial-time solvable. Under hard resource constraints, however, the problem has a different complexity, as analyzed below.

1) General Case:

Theorem 3. *Under hard resource constraints, the request scheduling problem (2) is generally NP-hard.*

Proof. We reduce the *partition problem* to our problem (2). Given a set of positive integers $A = \{t_1, \ldots, t_m\}$, the partition problem is the task of deciding whether A can be partitioned into two subsets A_1 and A_2, such that $\sum_{t_i\in A_1} t_i = \sum_{t_j\in A_2} t_j$. This problem is known to be NP-complete.

We construct an equivalent instance of the request scheduling problem as follows. We construct two edge clouds n_1 and n_2, each having unlimited communication capacity, unlimited storage capacity, and a computation capacity of $W = \frac{1}{2}\sum_{t_i\in A} t_i$. For each $t_i \in A$, we construct a request for a service l_i with computation requirement $\omega_{l_i} = t_i$, submitted to edge cloud n_1. Suppose that each edge cloud hosts all the services $l_1, \ldots, l_m$, and is allowed to serve any request. For this instance, (2) reduces to ($[m] \triangleq \{1, \ldots, m\}$):

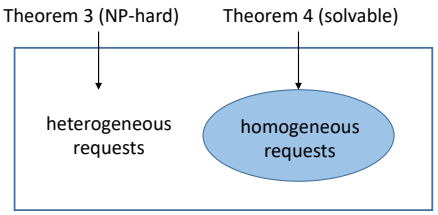


Figure 4. Complexity of request scheduling under hard constraints (2).

$$\max \sum_{i=1}^{m}\sum_{j=1}^{2} z_{l_i n_1 n_j} \tag{5a}$$

$$\text{s.t.} \sum_{j=1}^{2} z_{l_i n_1 n_j} \leq 1, \qquad \forall i \in [m], \tag{5b}$$

$$\sum_{i=1}^{m} t_i z_{l_i n_1 n_j} \leq W, \qquad \forall j \in [2], \tag{5c}$$

$$z_{l_i n_1 n_j} \in \{0, 1\}, \qquad \forall i \in [m], j \in [2]. \tag{5d}$$

Since $W = \frac{1}{2}\sum_{t_i\in A} t_i$, if A cannot be partitioned into two subsets of equal sum, then the sum for one of the subsets must be greater than W. This implies that we cannot serve every request while satisfying constraint (5c), and hence the optimal value of (5a) must be smaller than m. If A can be partitioned into subsets A_1 and A_2 of equal sum, then we must have $\sum_{t_i\in A_1} t_i = \sum_{t_i\in A_2} t_i = W$. Then setting $z_{l_i n_1 n_j} = 1$ if and only if $t_i \in A_j$ ($j = 1, 2$) gives a feasible solution to (5) with an objective value of m. Therefore, the partition problem has a solution if and only if all the requests can be served in the above instance of the request scheduling problem. □

Remark: The proof of Theorem 3 holds even if we require the edge clouds to be homogeneous, i.e., $K_n \equiv K$ and $W_n \equiv W$ ($\forall n \in N$). Thus, the request scheduling problem under hard resource constraints is NP-hard as long as the requests have heterogeneous resource demands.

2) Homogeneous Special Case: But what if the requests are homogeneous (i.e., $\kappa_l \equiv \kappa$, $\omega_l \equiv \omega$)? We show that the problem is no longer NP-hard in this case.

Theorem 4. *In the special case when all the requests have identical communication and computation demands, the scheduling problem (2) is polynomial-time solvable.*

We prove this theorem by developing a polynomial-time optimal solution in Section IV-C1.

3) Summary of All Cases: Together, Theorems 3 and 4 characterize the complexity of the request scheduling problem under hard constraints in all cases, as illustrated in Fig. 4.

IV. Algorithms

We now develop efficient algorithms for the service placement problem and the request scheduling problem separately.

A. Approximation Algorithm for Service Placement

Due to the NP-hardness of finding the optimal service placement in general (Section III-A), we seek efficient service placement algorithms with approximation guarantees.

1) Conversion to Set Function Optimization: We start by reformulating our problem as a set function optimization problem. Let $S \subseteq L \times N$ denote the set of selected single-service placements, where $(l,n) \in S$ means to place a replica of service l at edge cloud n. Let $\Omega(S)$ denote the optimal objective value of (1) for a fixed $\mathbf{x}$ given by $x_{ln} = 1$ if and only if $(l,n) \in S$. This can be calculated by solving the following (shadow) request scheduling problem, where $\mathbb{1}_{l,m}$ is the indicator function:

$$\max \sum_{l\in L}\sum_{n\in N}\lambda_{ln}\sum_{m\in N} y_{lnm} \tag{6a}$$

$$\text{s.t. } (1b),(1d),(1e), \tag{6b}$$

$$y_{lnm} \le a_{lnm}\mathbb{1}_{(l,m)\in S}, \qquad \forall l\in L, n\in N, m\in N, \tag{6c}$$

$$y_{lnm} \in [0,1], \qquad \forall l\in L, n\in N, m\in N. \tag{6d}$$

After that, we can rewrite the service placement problem as:

$$\max\ \Omega(S) \tag{7a}$$

$$\text{s.t.} \sum_{l:(l,n)\in S} r_l \le R_n, \qquad \forall n\in N, \tag{7b}$$

$$\sum_{(l,n)\in S} c_{ln} \le B, \tag{7c}$$

$$S \subseteq L\times N, \tag{7d}$$

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

First, we prove that, under certain conditions, the objective function of (7) has a desirable property.

Definition 1 ([47])**.** *A set function $f: 2^{\mathbf{x}} \to \mathcal{R}$ is monotone increasing if $\forall\ S_1 \subseteq S_2 \subseteq \mathbf{x}$, $f(S_1) \le f(S_2)$. Moreover, the function $f(.)$ is sub-modular if $\forall S_1 \subseteq S_2 \subseteq \mathbf{x}$ and $e \in \mathbf{x}\backslash S_2$, $f(\{e\}\cup S_1) - f(S_1) \ge f(\{e\}\cup S_2) - f(S_2)$.*

The significance of these properties is that if our objective function is monotone and sub-modular, then it is known [47] that the generic greedy algorithm can achieve a guaranteed approximation to the optimal solution.

Lemma 2. *The objective function $\Omega(S)$ in (7a) is a monotone sub-modular function for all feasible S if $\kappa_l \equiv \kappa$ ($\forall l \in L$), and*

1) *$\lfloor R_n r_l \rfloor \le 1$ for all $n \in N$ and $l \in L$, or*
2) *$W_m \ge \sum_{l\in L}\omega_l \sum_{n\in N}\lambda_{ln}$ for all $m \in N$.*

Proof. It is easy to see that $\Omega(S)$ is monotone, as expanding S will relax the constraint (6c), hence enlarge the solution space for (6) and increase its optimal objective value.

To show that $\Omega(S)$ is sub-modular, we need to show that for any sets $S_1, S_2 \subseteq L\times N$ and any $(l_1,n_1) \in (L\times N)\setminus S_2$, such that $S_1 \subseteq S_2$ and $S_2 \cup \{(l_1,n_1)\}$ is feasible, the following relationship holds

$$\Omega(S_1\cup\{(l_1,n_1)\}) - \Omega(S_1) \ge \Omega(S_2\cup\{(l_1,n_1)\}) - \Omega(S_2). \tag{8}$$

Suppose that $\mathbf{y}^{(0)}$ and $\mathbf{y}^{(2)}$ are the optimal scheduling solutions according to (6) under service placements S_1 and S_2, respectively. Moreover, suppose that $\mathbf{y}^{(1)}$ and $\mathbf{y}^{(3)}$ are the optimal scheduling solutions under service placements $S_1\cup\{(l_1,n_1)\}$ and $S_2\cup\{(l_1,n_1)\}$, respectively, that minimize the request rate scheduled to the replica (l_1,n_1), i.e., minimizing $\sum_{n\in N}\lambda_{l_1n}y_{l_1nn_1}$. We can then decompose the objective function as:

$$\Omega(S_1) = \sum_{(l,m)\in S_1}\sum_{n\in N}\lambda_{ln}y^{(0)}_{lnm}, \tag{9}$$

$$\Omega(S_1\cup\{(l_1,n_1)\}) = \sum_{(l,m)\in S_1}\sum_{n\in N}\lambda_{ln}y^{(1)}_{lnm} + \sum_{n\in N}\lambda_{l_1n}y^{(1)}_{l_1nn_1}, \tag{10}$$

$$\Omega(S_2) = \sum_{(l,m)\in S_2}\sum_{n\in N}\lambda_{ln}y^{(2)}_{lnm}, \tag{11}$$

$$\Omega(S_2\cup\{(l_1,n_1)\}) = \sum_{(l,m)\in S_2}\sum_{n\in N}\lambda_{ln}y^{(3)}_{lnm} + \sum_{n\in N}\lambda_{l_1n}y^{(3)}_{l_1nn_1}. \tag{12}$$

Due to this decomposition, we have

$$\text{LHS of (8)} = \sum_{(l,m)\in S_1}\sum_{n\in N}\lambda_{ln}(y^{(1)}_{lnm} - y^{(0)}_{lnm}) + \sum_{n\in N}\lambda_{l_1n}y^{(1)}_{l_1nn_1}, \tag{13}$$

$$\text{RHS of (8)} = \sum_{(l,m)\in S_2}\sum_{n\in N}\lambda_{ln}(y^{(3)}_{lnm} - y^{(2)}_{lnm}) + \sum_{n\in N}\lambda_{l_1n}y^{(3)}_{l_1nn_1}. \tag{14}$$

The first term in (13) is the difference in the request rate served by replicas in S_1 after/before placing the replica (l_1,n_1). Under condition (1) or (2) in the lemma, there is no contention of computation resources between replicas, and hence replicas in S_1 can still process requests scheduled to them under $\mathbf{y}^{(0)}$. Meanwhile, as the communication demands κ_l are the same for all types of requests, dropping requests originally scheduled to S_1 to admit requests to be scheduled to (l_1,n_1) will not improve the objective value of (6). Thus, the first term in (13) is zero. Similarly, the first term in (14) is also zero. The second term in (13,14) is the minimum request rate served by the replica (l_1,n_1) under an optimal scheduling, in the presence of replicas S_1 and S_2, respectively. Again, as there is no computation resource contention between replicas, requests that used to be served by replicas in S_1 under service placement $S_1\cup\{(l_1,n_1)\}$ can still be served there after adding replicas in $S_2\setminus S_1$, but these added replicas may offload some requests that used to be served by the replica (l_1,n_1). Therefore, $\sum_{n\in N}\lambda_{l_1n}y^{(1)}_{l_1nn_1} \ge \sum_{n\in N}\lambda_{l_1n}y^{(3)}_{l_1nn_1}$. This proves (8) and hence the sub-modularity of $\Omega(S)$. □

The constraints of (7) also have a desirable property.

Definition 2 ([48])**.** *Let X be a universe of elements. Consider a collection $\mathcal{I} \subseteq 2^X$ of subsets of X. $(X,\mathcal{I})$ is called an independence system if: (a) $\emptyset \in \mathcal{I}$, and (b) if $Z \in \mathcal{I}$ and $Y \subseteq Z$, then $Y \in \mathcal{I}$ as well. The subsets in $\mathcal{I}$ are called independent; for any set S of elements, an inclusion-wise maximal subset T of S that is in $\mathcal{I}$ is called a basis of S.*

Definition 3 ([48])**.** *Given an independence system $(X,\mathcal{I})$ and a subset $S \subseteq X$, the rank $r(S)$ is defined as the cardinality of the largest basis of S, and the lower rank $\rho(S)$ is the cardinality of the smallest basis of S. The independence system is called a p-independence system (or a p-system) if $\max_{S\subseteq X}\frac{r(S)}{\rho(S)} \le p$.*

Lemma 3. *The constraints (7b)-(7d) form a p-independence system for p= $\left\lceil \frac{\max\ c_{ln}}{\min_{c_{ln}>0}\ c_{ln}} \right\rceil + \left\lceil \frac{\max\ r_l}{\min_{l:r_l>0}\ r_l} \right\rceil$.*

Proof. By Definition 1, $(L\times N, \mathcal{I})$, where $\mathcal{I} \subseteq 2^{L\times N}$ is a set of all feasible solutions to (7) is an independent system, as $S = \emptyset$ is a feasible service placement, and the subset of any feasible service placement remains feasible. Consider any $S \subseteq L\times N$ and any two maximal feasible service placements $S_1,\ S_2 \subseteq S_1$. To add a pair $(l,n) \in S_2\backslash S_1$ to S_1, we need to

Algorithm 1: Greedy Service Placement based on Shadow Scheduling (GSP-SS)

Input: Input parameters of (1)

Output: Service placement $\mathbf{x} \triangleq (x_{lm})_{l\in L, m\in N}$

$S \leftarrow \emptyset$;

while $\exists\ (l,n) \in (L \times N)\ \backslash S$ such that $S \cup \{(l,n)\}$ satisfies (7b)-(7d) **do**

$(l^*, n^*) \leftarrow \arg\max_{(l,n):\ S\cup\{(l,n)\} \text{ satisfies (7b)-(7d)}} \Omega(S \cup \{(l,n)\})$;

$S \leftarrow S \cup \{(l^*, n^*)\}$;

Convert S to its vector representation $\mathbf{x}$;

take out a set S' of pairs form S_1, such that $(S1\backslash S') \cup \{(l,n)\}$ remains a feasible service placement. The set S' contains at most $\lceil \frac{\max\ r_l}{\min_{l:r_l>0}\ r_l} \rceil$ pairs from $\{l\} \times N$ corresponding to removing service replicas from edge cloud n to satisfy (7b), and at most $\lceil \frac{\max\ c_{ln}}{\min_{c_{ln}>0}\ c_{ln}} \rceil$ other pairs that correspond to removing service replicas with non-zero placement costs to satisfy (7c). Note that in the worst case all the existing service replicas under S_1 at edge cloud n have zero placement cost, and hence we need to remove replicas at other edge clouds to satisfy the budget constraint (7c). Repeating this swap to each pair in $S_2\backslash S_1$ shows that we reduce the number of placed service replicas by at most p-fold in modifying S_1 into S_2. Since the above holds for any $S \subset L \times N$ and any maximal independent subsets of S, the constraints (7b)-(7d) form a p-independent system. □

Combining Lemmas 2 and 3 gives the following result.

Theorem 5. *Under the conditions in Lemma 2, greedily optimizing (7) (Algorithm 1) yields a* $1/(1+p)$*-approximation for (1), where* $p = \lceil \frac{\max\ c_{ln}}{\min_{c_{ln}>0}\ c_{ln}} \rceil + \lceil \frac{\max\ r_l}{\min_{l:r_l>0}\ r_l} \rceil$.

Proof. From [47], for maximizing a monotone sub-modular function subject to a p-system constraint, the greedy algorithm has an approximation ratio of $1/(1+p)$. □

2) Algorithm: Algorithm 1 gives the pseudo code for the greedy algorithm. Note that it differs from the simple greedy heuristic in that each evaluation of $\Omega(\cdot)$ requires solving an instance of the shadow scheduling problem (6) using LP.

3) Complexity: There are $O(|N| \times \frac{R_{max}}{r_{min}})$ iterations in Algorithm 1, where $R_{max} = \max_{n\in N} R_n$ and $r_{min} = \min_{l\in L: r_l>0}\ r_l$. For each iteration, the algorithm considers $O(|L| \times |N|)$ single service placements, and for each single service placement, we need to evaluate the objective function by solving an $O(|L| \times |N|^2)$-variable $O(|L| \times |N|^2)$-bit input LP, which takes $O(|N|^{11} \times |L|^{5.5})$ time [49]. Therefore, the overall complexity of Algorithm 1 is $O(|N|^{13} \times |L|^{6.5} \times \frac{R_{max}}{r_{min}}) = O(|N|^{13} \times |L|^{6.5})$. We expect this complexity to be acceptable in practice, as this algorithm is only run once per frame, and the frame length will be at the scale of changes in request rates (usually tens of minutes or longer).

B. Optimal Request Scheduling under Soft Constraints

Given the number of requests of each type observed at the beginning of a slot and the service placement determined at the beginning of the current frame, the request scheduling problem under soft constraints is identical to (6), which can be solved by a generic LP solver in $O(|N|^{11} \times |L|^{5.5})$ time (see complexity analysis for Algorithm 1). We then use the resulting y_{lnm} to perform probabilistic scheduling, where a type-(l,n) request will be scheduled to each edge cloud $m \in N$ with probability y_{lnm}.

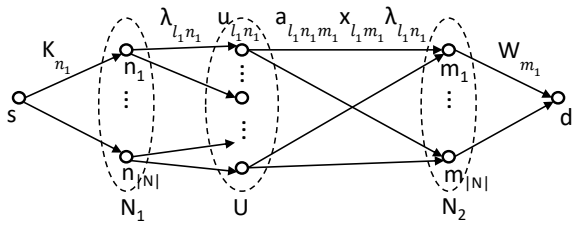


Figure 5. Auxiliary graph $\mathcal{G}$ for request scheduling.

C. Optimal and Heuristic Request Scheduling under Hard Constraints

1) Optimal Algorithm for Homogeneous Requests: Consider the special case where all the requests have identical communication and computation demands. Without loss of generality[3], assume $\kappa_l \equiv 1$, $\omega_l \equiv 1$, and K_n and W_n are integers for all $n \in N$. We will show that in this special case, (2) can be converted to a maximum flow problem in an auxiliary graph, and is thus polynomial-time solvable by existing maximum flow algorithms.

Graph construction: Given parameters of (2) (including the service placement x_{lm} of the current frame), we construct an auxiliary graph $\mathcal{G}$ as in Fig. 5. The nodes in $\mathcal{G}$ consist of a source s, a destination d, a set of nodes U in 1-1 correspondence with the types of requests $\{(l,n)\}_{l\in L, n\in N}$, 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 $n \in N_1$ by a directed link of capacity K_n, and each node $m \in N_2$ is connected to node d by a directed link of capacity W_m. Moreover, each node $n \in N_1$ is connected to each node $u_{ln} \in U$ (representing type-(l,n) requests) by a directed link of capacity λ_{ln}, and each node $u_{ln} \in U$ is connected to each node $m \in N_2$ by a directed link of capacity $a_{lnm}x_{lm}\lambda_{ln}$.

Conversion to a maximum flow problem: We will show that for homogeneous requests, the problem of request scheduling under hard resource constraints is equivalent to a maximum flow problem in $\mathcal{G}$.

Theorem 6. *For homogeneous requests, the optimal value of (2) equals the maximum flow between* s *and* d *in* $\mathcal{G}$*, and the optimal solution is to set* z_{lnm} *to the flow rate on link* (u_{ln}, m) *under the maximum integral flow from* s *to* d*.*

Proof. First, an s-to-d flow satisfies the link capacities in $\mathcal{G}$ if and only if the corresponding $\mathbf{z} = (z_{lnm})_{l\in L, n,m\in N}$ satisfies constraints (2b)–(2e). This is because if the rate on link (u_{ln}, m) represents the number of type-(l,n) requests that are served by edge cloud m, then by flow conservation, the flow rate on link (s,n) represents the number of served requests that are submitted to edge cloud n, the rate on link (n, u_{ln}) represents the number of served requests of type (l,n), and the rate on link (m,d) represents the total number of requests served by edge cloud m. Thus, by construction, satisfying

[3]We redefine K_n as the maximum number of requests that an edge cloud can communicate with its covered users in a slot (for input/output), and W_n as the maximum number of requests that an edge cloud can process in a slot.

Algorithm 2: Maximum Flow-based Request Scheduling (MFRS)

input : Input parameters of (2), assuming $\kappa_l \equiv 1$, $\omega_l \equiv 1$, and K_n and W_n are integers ($\forall n \in N$)
output: Request scheduling $\mathbf{z} = (z_{lnm})_{l\in L,n,m\in N}$
1 $\mathcal{G} \leftarrow$ auxiliary graph as in Fig. 5;
2 compute the maximum integral flow from s to d in $\mathcal{G}$;
3 **foreach** $(l, n, m) \in L \times N \times N$ **do**
4 $\quad z_{lnm} \leftarrow$ flow rate on link (u_{ln}, m) in $\mathcal{G}$;

the capacities of these links is equivalent to satisfying the corresponding constraints in (2b)–(2e). If we impose a further integral flow constraint, i.e., the flow rate on every link must be an integer, then constraint (2f) is also satisfied. Moreover, by the *Integral Flow Theorem* [50], there exists an integral flow between s and d that achieves the maximum flow rate, as the link capacities in $\mathcal{G}$ are all integers. Thus, the optimal objective value of (2) equals the maximum integral s-to-d flow, which in turn equals the maximum s-to-d flow. □

Algorithm: By Theorem 6, we develop a scheduling algorithm called *Maximum Flow-based Request Scheduling (MFRS)*, shown in Algorithm 2[4]. We can leverage existing maximum flow algorithms to implement line 2. In particular, the Ford-Fulkerson algorithm [50] has guaranteed termination and optimality. More importantly, for a graph with integral link capacities, this algorithm gives an integral solution, i.e., only sending an integral amount of flow per link. The optimality of this algorithm is implied by Theorem 6.

Corollary 1. *For homogeneous requests, MFRS (Algorithm 2) maximizes the number of requests served by the edge clouds.*

Complexity: It is easy to see that constructing $\mathcal{G}$ (line 1) takes $O(|L| \cdot |N|^2)$ time, and converting the maximum flow solution to a scheduling solution (lines 3–4) also takes $O(|L| \cdot |N|^2)$ time. It is known that for integral link capacities, the Ford-Fulkerson algorithm has complexity $O(|E| \cdot \phi)$, where $|E|$ is the number of links and ϕ is the maximum flow. In our case, $|E| = O(|L||N|^2)$ and $\phi \le \min(\sum_{n\in N} K_n, \sum_{m\in N} W_m, \sum_{l\in L}\sum_{n\in N} \lambda_{ln})$. Therefore, the overall complexity of Algorithm 2 is $O(|L||N|^2 \min(\sum_{n\in N} K_n, \sum_{m\in N} W_m, \sum_{l\in L}\sum_{n\in N} \lambda_{ln}))$.

Remark: We note that Algorithm 2 extends our previous algorithm *Optimal Request Scheduling* in [2], which requires both the requests and the edge clouds to be homogeneous.

2) Heuristic Algorithm for Heterogeneous Requests: In the general case where requests for different services can have different communication/computation demands, we resort to LP relaxation, i.e., replacing the integer constraint (2f) by a linear constraint $z_{lnm} \ge 0$. The key is how to round the fractional solution to this LP relaxation to a feasible integral solution to (2). To this end, we propose *LP Relaxation-based Request Scheduling (LRRS)*, shown in Algorithm 3. LRRS sequentially rounds each fractional scheduling variable z'_{lnm} to the nearest integer while staying within the constraints of (2). This is achieved by maintaining the *residual* number of requests $\widetilde{\lambda}_{ln}$, the *residual* communication capacity $\widetilde{K}_n$, and the *residual* computation capacity $\widetilde{W}_m$. For each triple (l, n, m), $\min(a_{lnm}x_{lm}\widetilde{\lambda}_{ln}, \lfloor\frac{\widetilde{K}_n}{\kappa_l}\rfloor, \lfloor\frac{\widetilde{W}_m}{\omega_l}\rfloor)$ is the maximum number of type-(l, n) requests that can be scheduled to edge cloud m without violating any constraint or changing any existing scheduling decision. Thus, line 6 results in a best-effort approximation of the optimal fractional solution while enforcing the hard constraints of (2).

[4]Algorithm 2 is mainly of theoretical value, as it shows that the non-trivial ILP (2) when $\kappa_l \equiv 1$ and $\omega_l \equiv 1$ has a polynomial-time optimal solution.

Algorithm 3: LP Relaxation-based Request Scheduling (LRRS)

input : Input parameters of (2)
output: Request scheduling $\mathbf{z} = (z_{lnm})_{l\in L,n,m\in N}$
1 $\mathbf{z}' \leftarrow$ optimal solution to the LP relaxation of (2);
2 $\widetilde{\lambda}_{ln} \leftarrow \lambda_{ln}$ for all $l \in L, n \in N$;
3 $\widetilde{K}_n \leftarrow K_n$ for all $n \in N$;
4 $\widetilde{W}_m \leftarrow W_m$ for all $m \in N$;
5 **foreach** $(l, n, m) \in L \times N \times N$ **do**
6 $\quad z_{lnm} \leftarrow$ $\min\Big(\text{round}(z'_{lnm}),\ \min(a_{lnm}x_{lm}\widetilde{\lambda}_{ln}, \lfloor\frac{\widetilde{K}_n}{\kappa_l}\rfloor, \lfloor\frac{\widetilde{W}_m}{\omega_l}\rfloor)\Big)$;
7 $\quad \widetilde{\lambda}_{ln} \leftarrow \widetilde{\lambda}_{ln} - z_{lnm}$;
8 $\quad \widetilde{K}_n \leftarrow \widetilde{K}_n - \kappa_l \cdot z_{lnm}$;
9 $\quad \widetilde{W}_m \leftarrow \widetilde{W}_m - \omega_l \cdot z_{lnm}$;

Complexity: Line 1 of Algorithm 3 is the same as request scheduling under soft constraints, whose complexity is $O(|L|^{5.5} \times |N|^{11})$ (see Section IV-B). The rounding takes $O(|L| \times |N|^2)$ time, as there are $O(|L| \times |N|^2)$ iterations and each iteration (lines 6–9) takes a constant time. Thus, LRRS has a complexity of $O(|L|^{5.5} \times |N|^{11})$.

V. Extension to Multi-frame Optimization

So far we have only considered the optimizations within one frame, with the assumption that the solutions will be repeatedly applied in each frame. However, for recurrent workloads, it is possible to predict the request rates for a larger time window (e.g., 24 hours) that contains multiple frames, each being a time interval with constant request rates. In this case, the frame-by-frame optimization framework for service placement can incur sub-optimality, as it neglects the correlation across frames, in the sense that the cost of placing a replica of service l at a given edge cloud depends on where service l was placed in the previous frame. To capture the correlation, we need to jointly optimize the service placement across all the predictable frames.

Let F be the set of frames for which request rate prediction is available. Our objective is to maximize the expected number of requests served over all the frames:

$$\max \sum_{f\in F} T_f \sum_{l\in L}\sum_{n\in N} \lambda^f_{ln} \sum_{m\in N} y^f_{lnm} \tag{15a}$$

$$\text{s.t.} \sum_{m\in N} y^f_{lnm} \le 1, \qquad \forall l \in L, n \in N, f \in F, \tag{15b}$$

$$\sum_{l\in L} x^f_{lm} r_l \le R_m, \qquad \forall m \in N, f \in F, \tag{15c}$$

$$\sum_{l\in L} \lambda^f_{ln}\kappa_l \sum_{m\in N} y^f_{lnm} \le K_n, \qquad \forall n \in N, f \in F, \tag{15d}$$

$$\sum_{l\in L} \omega_l \sum_{n\in N} \lambda^f_{ln} y^f_{lnm} \le W_m, \qquad \forall m \in N, f \in F, \tag{15e}$$

$$y^f_{lnm} \le a_{lnm} x^f_{lm}, \qquad \forall l \in L, n, m \in N, f \in F, \tag{15f}$$

$$\sum_{l\in L}\sum_{n\in N} x^f_{ln} \min_{n'\in N_+} (c_{ln'n} x^{f-1}_{ln'} + c_{\max}(1 - x^{f-1}_{ln'})) \le B, \qquad \forall f \in F, \tag{15g}$$

$$x^f_{ln} \in \{0, 1\}, y^f_{lnm} \ge 0, \qquad \forall l \in L, n, m \in N, f \in F. \tag{15h}$$

This formulation is similar to the single-frame formulation (1), but the scope is extended to multiple frames. The real difference is the non-linear constraint (15g), where $c_{\max} > \max_{l,n',n} c_{ln'n}$ is a large constant. Essentially, $\min_{n' \in N_+}(c_{ln'n} x_{ln'}^{f-1} + c_{\max}(1 - x_{ln'}^{f-1}))$ is the minimum cost of placing a replica of service l at edge cloud n in frame f, which depends on the service placement in frame $f-1$.

The multi-frame optimization (15) is a *mixed integer non-linear program (MINP)* that is even harder than (1). Given a service placement $(x_{ln}^f)_{f \in F, l \in L, n \in N}$, the remaining optimization is still an LP in $(y_{lnm}^f)_{f \in F, l \in L, n,m \in N}$. Thus, GSP-SS (Algorithm 1) still applies, where we iteratively place one replica at a time in a selected frame, subject to constraints (15c, 15g), to maximize the objective value of the corresponding LP. We leave detailed analysis to future work.

VI. Performance Evaluation

We have evaluated the performance of the proposed algorithms using both synthetic and trace-driven simulations under soft and hard constraints. For the synthetic cases we show the impact of varying constraints and characteristics of service requests to show that our algorithms are robust. For the trace cases, we show how frame and slot sizes should be set to achieve desirable performance.

A. Benchmarks

To assess the performance of the proposed service placement algorithm, we use the following benchmarks:

1) *the optimal solution* of (1) using an MILP solver (MATLAB `intlinprog`);
2) *LP-relaxation with rounding*, which first solves the LP relaxation of (1), and then rounds the placement variables to $\{0, 1\}$, subject to R- and B-constraints;
3) *top-k service placement*, which sequentially considers each edge cloud $m \in N$, computes the total demand for each service l that can be scheduled to m, defined as $\Lambda_{lm} = \sum_{n \in N} \lambda_{ln} a_{lnm}$, and then places services at m in the descending order of Λ_{lm} until reaching R_m or exhausting the budget.

To assess the performance of the proposed request scheduling algorithm under hard constraints, we use the following benchmarks:

1) *the optimal solution* of (2) using an ILP solver (MATLAB `intlinprog`);
2) *greedy request scheduling*, which sequentially considers each triple $(l, n, m) \in L \times N \times N$ and schedules as many type-(l, n) requests to edge cloud m as possible, without violating any constraint in (2) or changing any existing scheduling decision.

We note that no benchmark is needed for request scheduling under soft constraints, as the problem is an LP and hence polynomial-time solvable.

B. Results on Service Placement

1) Synthetic simulation: For synthetic simulations, we show results under two settings.

Setting 1: First, we set $|N| = 6$ and $|L| = 100$. We initially draw the values for R_n, K_n and W_n ($\forall n \in N$) uniformly from the intervals $[24, 36]$, $[16, 24]$ and $[32, 48]$, respectively. We then set these values differently based on a representative set of applications. [5] Assuming that the edge clouds are associated with hexagon cells arranged into two rows, we set the costs of replicating a service from an edge cloud k hops away or the remote cloud to $0.2k$ and 2, respectively, and the budget B to $0.2 \cdot |N| \cdot |L|$. We set a_{lnm} such that each request can only be served by edge clouds within 2 hops of the edge cloud it is submitted to. The arrival rate of request l is obtained as $\lambda_{ln} = \lambda_n p_{ln}$, where λ_n (total request rate in edge cloud n) is drawn randomly from the interval $[3, 5]$. For p_{ln} (popularity of service l in edge cloud n), we draw a random subset of services L_n, and set $p_{ln} \propto i_l^{-\alpha}$ for each $l \in L_n$, where i_l is the rank of l in L_n, and $\alpha = 0.5$ is the skewness parameter of Zipf's distribution. We initialize the system by randomly placing $|L|/8$ services. For each service l, κ_l, ω_l and r_l are drawn uniformly from $[0.5, 1]$. All results are averaged over 50 Monte Carlo runs. In the above settings, we set communication resource to be the most restrictive, followed by storage and then computation. This is to model the resource demands of data-intensive applications. Besides the above default settings, we will also explore the parameter space by varying these parameters one by one in order to evaluate the impact of each parameter.

Figs. 6 (a-d) illustrate the effect of increasing various resource parameters on the percentage of served requests, including the computation capacity W_n, the service placement budget B, the storage capacity R_n, and the communication capacity K_n. As expected, an increase in the resource capacities leads to a higher percentage of served requests. This trend is more obvious in Figs. 6 (b-c), as these resources directly affect the set of feasible service placements. When comparing the performance of different algorithms under the same resource capacities, we observe that GSP-SS considerably outperforms LP-relaxation with rounding and top-k. Furthermore, it is very close to the optimal solution. We have verified that GSP-SS achieves over 90% of the optimal performance, i.e., the ratio of served requests when using GSP-SS versus the optimal solution is greater than 0.9 on the average. Similar observations have been made in the other simulations as well.

We further vary parameters of request generation. Fig. 6 (e) shows that as we increase the average request rate, the percentage of served requests decreases notably due to the contention of resources. Fig. 6 (f) shows that as we increase the skewness of service popularities (by increasing α), the optimal solution and GSP-SS remain the same, while the baselines (LP relaxation with rounding and top-k) improve slightly. This is because the increased skewness causes the requests to be more and more concentrated on a few popular services, making service placement easier.

Finally, we vary the number of edge clouds $|N|$ in Fig. 6 (g). As expected, the more edge clouds, the more resources, and hence the performance improves. However, we see from Figs. 6 (a-d) that similar improvements can be achieved by increasing the capacities of existing edge clouds or the service placement budget. This shows the effect of resource pooling.

We note that our simulation setup does not satisfy the condition in Theorem 5 as the κ_l's are different, and thus

[5] The values of κ_l and K_n are in KBps, r_l and R_n in TB, and ω_l as well as W_n in Mflops/s.

the theoretical approximation guarantee does not apply. Nevertheless, we have observed empirically that GSP-SS always yields near-optimal performance.

Setting 2: Next, we consider scenarios in which the values of the input parameters are at the same order of magnitude as those corresponding to specific highly popular data-intensive services [51], [52], i.e., video analytics and ultra-reliable virtual reality. The storage requirement for the services is chosen uniformly from the interval $[0.5, 1]$ TB, bandwidth requirement is uniform in $[5, 10]$ Mbps and computation requirement is uniform in $[50, 100]$ Mflops/s.

Figs. 7 (a-d) depict the percentage of served request for different values of communication capacity, arrival rate, storage capacity, and computation capacity. The communication capacity of the edge clouds is chosen uniformly on $[20, 30]$ Mbps, computation capacity is uniform on $[320, 480]$ Mflops/s, whereas storage capacity is chosen uniformly from the range $[24, 36]$ TB. The capacities of edge clouds are highly heterogeneous depending on the cost and the targeted services. The above parameter setting targets at services requiring large amounts of server data; later in trace-driven simulation, we will set the parameters to target at services requiring large amounts of computation. We see that the trend and the comparison between different algorithms are qualitatively the same as in Fig. 6. In particular, the proposed algorithm GSP-SS performs nearly as well as the optimal and notably better than the benchmarks of LP-relaxation with rounding and top-k.

2) Trace-driven simulation: We cross-validate our observations in a more realistic scenario driven by traces. For the trace-driven simulation, we extract user and edge cloud locations from real mobility traces and cell tower locations. We use the *taxicab* traces from [53], by extracting the traces of 36 users over a 520-minute period with location updates every 1 minute. We assign users into Voronoi cells based on cell tower locations obtained from `http://www.antennasearch.com`, from which we select a subset of 6 cell towers that are at least 9.5 km apart to represent the locations of edge clouds.

User requests are generated from a wireless trace from [54], containing transmission timestamps generated by 5 different applications from 36 wireless devices. We associate each device with a user in the taxicab trace, and duplicate each trace 5 times to obtain $|L| = 25$ services. As each timestamp in the original trace represents a single packet, we stretch the time axis by 60 (by treating the time unit as 'minute' instead of 'second') to simulate the arrival process of service requests. The obtained request rates range from 288,935 to 650,415, with a mean of 521,070 (requests/slot).

For each edge cloud, we randomly choose a storage capacity R_n of 3-6 TB, a communication capacity of 16-48 Mbps (i.e., $K_n \in [0.12, 0.36]$ GB/slot), and a computation capacity of 50-100 Gflops/s (i.e., $W_n \in [3, 6]$ Tflops/slot), unless stated otherwise.

The other parameters are as before, except that the units of κ_l and ω_l change to GB/slot and Tflops/slot.

Setting Frame and Slot duration: The frame and slot durations are engineering decisions based on the characteristics of the system and desired performance. The frame duration is the time during which a deployment of services on servers remains constant. The re-deployment of services has a cost which in our system is constrained by budget B. How often to move services is driven by how stable the system is in terms of user mobility and user request characteristics (e.g., rate, resource requirements). A system operator will know the expected dynamics of a system based on trends determined over time. As trends change, the duration of the frames can change.

Slot duration is set based on the scheduling delay and job execution time in a system. In our system, jobs are expected to complete their processing within one slot. Therefore, slot duration should be set as small as possible so that jobs can complete so that scheduling delays are small.

To set the frame duration for the trace evaluation, we plotted the performance of the system for different frame and slot durations. Fig. 8 illustrates the effect of varying the frame size. In this scenario, the slot duration is 1 min. Fig. 8 depicts the percentage of served requests vs. frame size. As can be seen from Fig. 8, the performance starts deteriorating considerably for frames that are longer than 30 slots. Choosing a shorter frame provides only a slightly higher percentage of served requests, but increases the cost. Therefore we choose a frame duration of 30 minutes (slots in this case). 'Predicted' values are the predicted percentage of served requests when solving (1) at the beginning of each frame. 'Actual' values are the actual percentage of requests served in each slot under soft constraints, obtained by solving (6) for the requests arrived in that slot and the service placement of the corresponding frame.

Next, we look at the impact of the slot duration on the performance. We consider a frame length of 30 minutes, with 5 different slot durations: 1, 2, 3, 5 and 10 minutes. This means that the frames consist of 30, 15, 10, 6 and 3 slots, respectively. The values of K and W are scaled with the duration of the slot so overall system resources are kept constant. Fig. 9 illustrates the percentage of served requests vs. slot duration. As can be seen from Fig. 9, the performance is almost completely insensitive to the slot duration. Since we are interested in providing the shortest possible delay, we choose the slot duration to be 1 minute.

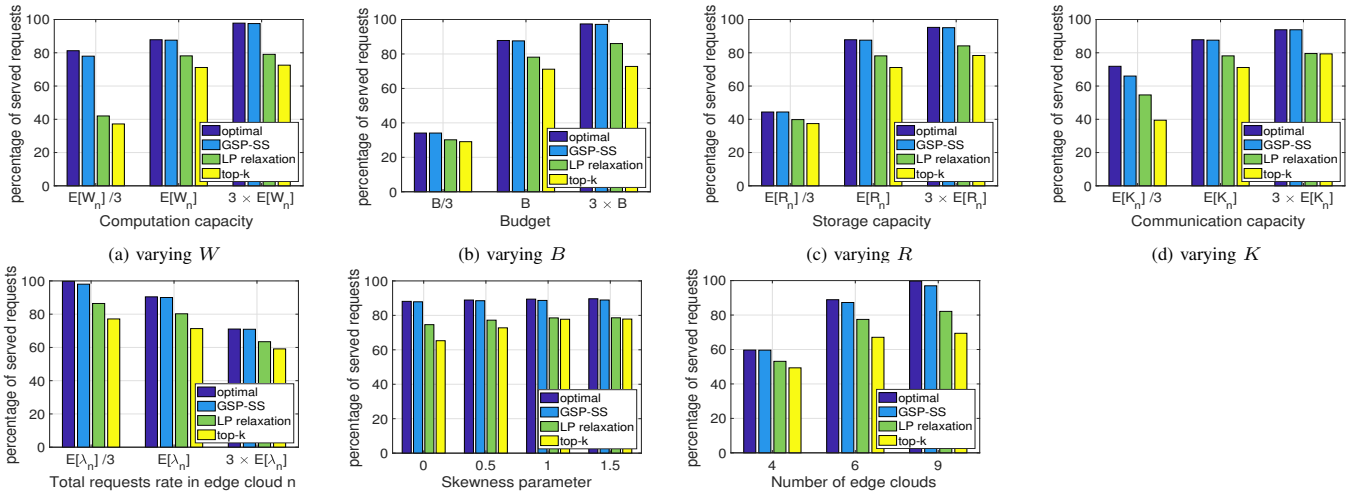
In this set of experiments we use a frame duration of 30 minutes and slot duration of 1 minute as described above.

Results: Fig. 10 shows the performance of each algorithm over time. 'Predicted' and 'actual' values have the same meaning as in Fig. 8. Fig. 10 shows that GSP-SS closely approximates the optimal not only in the predicted performance but also in the actual performance, while outperforming the baselines.

C. Results on Request Scheduling

1) Soft Constraints: Fig. 10 ('actual') already shows the performance achieved by probabilistic scheduling under soft resource constraints. Since the optimal probabilistic schedule is not hard to compute (by solving (6)), the focus here is to understand to what extent this probabilistic schedule adheres to the resource constraints.[6]

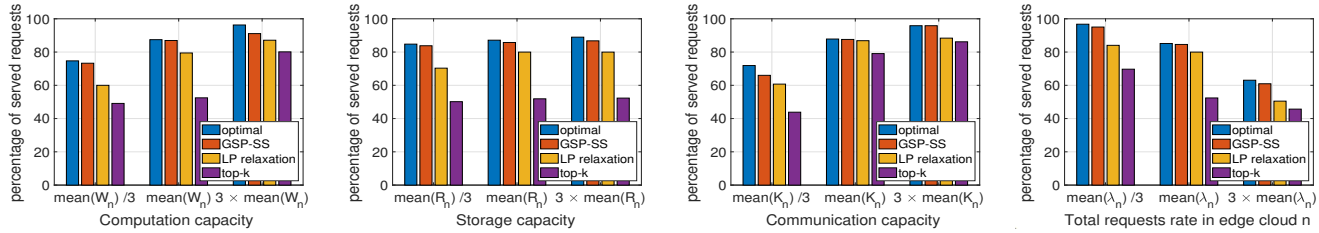
[6]It is worth mentioning that the algorithm based on soft constraints is only designed for delay-tolerant applications, in which case our results show that it does not cause frequent capacity violations or huge load spikes. In the case of applications with stringent SLAs, we have proposed a set of different algorithms based on hard constraints, which guarantee that SLAs will be satisfied (while making a best effort in serving requests at the edge).



(a) varying W (b) varying B (c) varying R (d) varying K

(e) varying λ (f) varying α (g) varying $|N|$

Figure 6. Performance evaluation for service placement under the synthetic simulation setup in Section VI-B.



(a) varying W (b) varying R (c) varying K (d) varying λ

Figure 7. Performance evaluation for service placement under highly popular data-intensive service input.

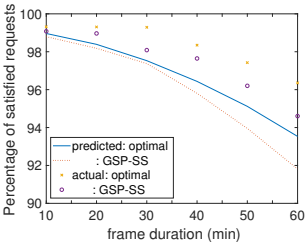


Figure 8. Varying the frame size.

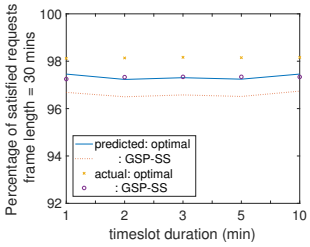


Figure 9. Varying the slot duration.

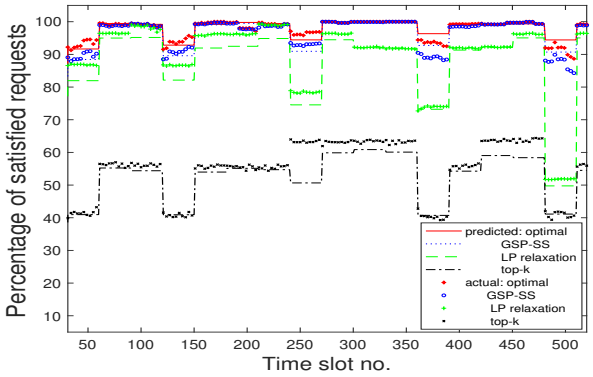


Figure 10. Performance evaluation in trace-driven simulation under soft constraints.

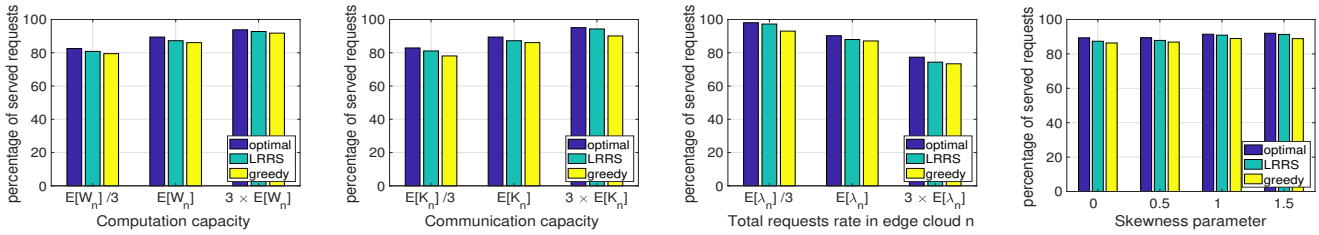
Given that probabilistic scheduling only satisfies the K-constraint (1d) and the W-constraint (1e) *on the average*, it is possible that the scheduled requests temporarily exceed the communication/computation capacity of an edge cloud. To understand the extent of capacity violations, we evaluate both the frequency and the severity of capacity violations. Table II shows the results for each type of resource (computation/communication) at each edge cloud. These results are obtained under the default parameter setting for synthetic simulations specified in Section VI-B. As can be seen from Table II, there are capacity violations in about 5% of the slots, and in these slots, the amount by which the capacities are exceeded is about 10%. These are moderate violations, which justifies the use of probabilistic scheduling and soft resource constraints for services that are not highly delay-sensitive.

Table II
CAPACITY VIOLATIONS (%)

Edge cloud #	1	2	3	4	5	6
% of time K violated	6.33	4.64	2.85	2.86	4.66	2.28
% of time W violated	1.29	1.65	3.26	4.15	3.59	0.95
Amount K violated	3.05	11.48	2.47	7.68	5.61	7.84
Amount W violated	10.69	0.72	9.61	2.87	0.51	2.90

2) Hard Constraints: Similar to Fig. 6, we use synthetic simulations to evaluate the impacts of different input parameters for request scheduling under hard resource constraints. All the results are obtained under the optimal service placement.

The results are shown in Fig. 11, where (a) shows the impact of increasing the computation capacity (W), (b) shows the impact of increasing the communication capacity (K), (c) shows the impact of increasing the rate of requests (λ), and (d) shows the impact of increasing the skewness parameter (α). In every plot, we compare the proposed LRRS algorithm (Algorithm 3) with the optimal solution and the greedy solution that are explained in Section VI-A. Note that MFRS (Algorithm 2) requires $\kappa_l = \omega_l = 1$ for all $l \in L$, which is not satisfied here. While the impacts of these parameters are similar to



(a) varying W (b) varying K (c) varying λ (d) varying α

Figure 11. Performance evaluation for request scheduling under hard constraints.

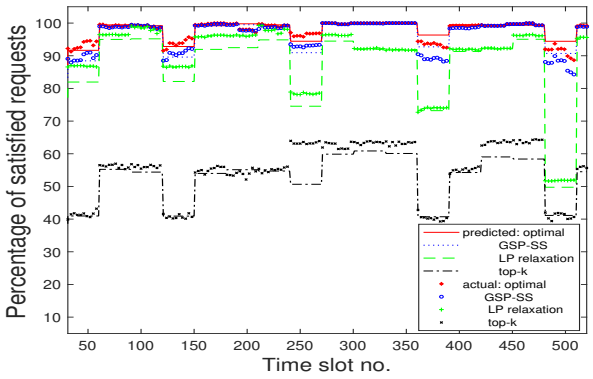


Figure 12. Performance evaluation in trace-driven simulation under hard constraints (scheduling done by LRRS).

those observed in Fig. 6, these results differ from Fig. 6 in that there is not much difference between different algorithms. In particular, the greedy request scheduling already approximates the optimal request scheduling, and LRRS performs in between. This observation indicates that the key performance differentiator is the service placement algorithm, and it suffices to use a simple algorithm for request scheduling.

We now show results for hard constraints using the same trace inputs as described above. As in Fig. 10, we show in Fig. 12 both the 'predicted' percentage of served requests, computed at the beginning of each frame based on the predicted request rates, and the 'actual' percentage of served requests in each slot. The difference from Fig. 10 is that the 'actual' values are computed by LRRS under hard resource constraints (while the 'predicted' values are the same as in Fig. 10). Comparing Fig. 10 and 12, we see that imposing hard resource constraints *does not* significantly reduce the percentage of served requests, while having the advantage that every scheduled request is guaranteed to finish within one slot. This justifies the use of deterministic scheduling and hard resource constraints for real-time services.

D. Results on Multi-frame Extension

Finally, we evaluate the extended GSP-SS for the multi-frame optimization (15). Fig. 13 shows the performance based on request prediction over an $|F|$-frame sliding window. We skip the other algorithms in this evaluation, as top-k performs the same as in Fig. 10, the optimal solution is hard to compute due to nonlinearity of (15), and LP relaxation does not apply. As in Fig. 10, 'predicted' values are based on the request rates predicted at the beginning of each window, and 'actual' values are based on the actual request rates in each slot. We see that prediction over a larger window improves the performance of GSP-SS in terms of the actual values, and 2-frame prediction appears to be sufficient.

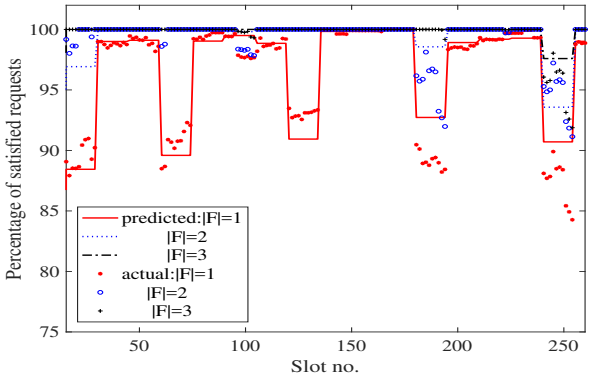


Figure 13. Performance of extended GSP-SS for multi-frame optimization in trace-driven simulation.

Although $|F| = 2$ suffices in this evaluation, we have observed that if we increase the resource contention, a larger prediction window can help. For example, if we double the request rates, then the predicted (actual) percentage of served requests will become 73.63% (72.11%) for $|F| = 1$, 74.37% (72.90%) for $|F| = 2$, and 79.11% (78.50%) for $|F| = 3$.

VII. Conclusion

We proposed a two-time-scale solution for joint service placement and request scheduling in a system of networked edge clouds under communication, computation, and storage constraints. We not only proved the NP-hardness of the problem in the general case, but also characterized its complexity in all the special cases. By combining the greedy heuristic with shadow request scheduling, we developed a polynomial-time service placement algorithm, which was proved to give a constant approximation ratio under certain conditions. We further showed that the problem of request scheduling under hard resource constraints, although NP-hard in general, can be solved in polynomial time if all the requests demand the same amounts of communication and computation resources, in which case we developed a polynomial-time optimal solution based on the maximum flow algorithm. Extensive simulations showed that the key performance differentiator is the service placement algorithm, and the proposed service placement algorithm achieves near-optimal performance. As part of the future work, we plan to consider reinforcement learning techniques for request predictions. Also, we are planning to consider a system that is capable of supporting ultra-high reliability and low latency (URLL) services.

REFERENCES

[1] V. Farhadi, F. Mehmeti, T. He, T. L. Porta, H. Khamfroush, S. Wang, and K. S. Chan, "Service placement and request scheduling for data-intensive applications in edge clouds," in *IEEE INFOCOM*, April 2019.

[2] 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," in *IEEE ICDCS*, July 2018.

[3] 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.

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

[5] 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.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *MCC*, 2012.

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

[8] 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," *IEEE Tran. Parallel and Dist. Syst.*, vol. 28, no. 4, 2017.

[9] 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

[10] 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 SEC*, 2017.

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

[12] 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.

[13] T. Taleb, A. Ksentini, and P. Frangoudis, "Follow-me cloud: When cloud services follow mobile users," *IEEE Tran. Cloud Comput.*, 2018.

[14] 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*, 2015.

[15] 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.

[16] A. Ceselli, M. Premoli, and S. Secci, "Mobile edge cloud network design optimization," *IEEE/ACM Trans. on Netw.*, vol. 25, no. 3, 2017.

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

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

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

[20] L. Wang, L. Jiao, J. Li, and M. Muhlhauser, "Online resource allocation for arbitrary user mobility in distributed edge clouds," in *IEEE ICDCS*, 2017.

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

[22] 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.

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

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

[25] 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.

[26] I.-H. Hou, T. Zhao, S. Wang, and K. Chan, "Asymptotically optimal algorithm for online reconfiguration of edge-clouds," in *ACM MobiHoc*, 2016.

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

[28] 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, June 2017.

[29] A. Aral and T. Ovatman, "A decentralized replica placement algorithm for edge computing," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 516–529, June 2018.

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

[31] 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.

[32] 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.

[33] 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.

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

[35] A. A. Haghighi, S. S. Heydari, and S. Shahbazpanahi, "Dynamic QoS-aware resource assignment in cloud-based content-delivery networks," *IEEE Access*, vol. 6, pp. 2298–2309, 2018.

[36] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE INFOCOM*, April 2019.

[37] Yazhou Hu, Bo Deng, Fuyang Peng, and Dongxia Wang, "Workload prediction for cloud computing elasticity mechanism," in *Proc. IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, 2016.

[38] G. Kecskemeti, Z. Nemeth, A. Kertesz, and R. Ranjan, "Cloud workload prediction based on workflow execution time discrepancies," *Cluster Computing*, vol. 22, no. 3, 2019.

[39] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *Proc. IEEE Network Operations and Management Symposium*, 2012.

[40] D. Bertsimas and J. Tsitsiklis, *Introduction to linear optimization*. Athena Scientific, 1997.

[41] M. Conforti, G. Cornuejols, and G. Zambelli, *Integer programming*. Springer, 2014.

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

[43] 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.

[44] 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.

[45] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack problems*. Springer, 2004.

[46] M. Cardei and D.-Z. Du, "Improving wireless sensor network lifetime through power aware organization," *Wireless Networks*, vol. 11, no. 3, p. 333–340, 2005.

[47] 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.

[48] A. Gupta, A. Roth, G. Schoenebeck, and K.Talwar, "Constrained non-monotone submodular maximization: Offline and secretary algorithms," *Lecture Notes in Computer Science (LNCS)*, vol. 6484, no. 12, 2010.

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

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

[51] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler, "Toward low-latency and ultra-reliable virtual reality," *IEEE Network*, vol. 32, no. 2, 2018. [Online]. Available: https://doi.org/10.1109/MNET.2018.1700268

[52] Z. Lu, K. Chan, R. Urgaonkar, and T. L. Porta, "On-demand video processing in wireless networks," in *Proc. IEEE ICNP*, 2016.

[53] 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

[54] A. S. Uluagac, "CRAWDAD dataset gatech/fingerprinting (v.2014-06-09)," Downloaded from https://crawdad.org/gatech/fingerprinting/20140609/isolatedtestbed, Jun. 2014, traceset: isolatedtestbed.