Approximation Algorithms for Data-Intensive Service Chain Embedding

Konstantinos Poularakis
Yale University
konstantinos.poularakis@yale.edu

Jaime Llorca
New York University
jllorca@nyu.edu

Antonia M. Tulino
University of Napoli Federico II & New York University
antoniamaria.tulino@unina.it

Leandros Tassiulas
Yale University
leandros.tassiulas@yale.edu

ABSTRACT

Recent advances in network virtualization and programmability enable innovative service models such as Service Chaining (SC), where flows can be steered through a pre-defined sequence of service functions deployed at different cloud locations. A key aspect dictating the performance and efficiency of a SC is its instantiation onto the physical infrastructure. While existing SC Embedding (SCE) algorithms can effectively address the instantiation of SCs consuming computation and communication resources, they lack efficient mechanisms to handle the increasing data-intensive nature of next-generation services. Differently from computation and communication resources, which are allocated in a *dedicated* per request manner, storage resources can be *shared* to satisfy multiple requests for the same data. To fill this gap, in this paper, we formulate the data-intensive SCE problem with the goal of minimizing storage, computation, and communication resource costs subject to resource capacity, service chaining, and data sharing constraints. Using a randomized rounding technique that exploits a novel data-aware linear programming decomposition procedure, we develop a multi-criteria approximation algorithm with provable performance guarantees. Evaluation results show that the proposed algorithm achieves near-optimal resource costs with up to 27.8% of the cost savings owed to the sharing of the data.

CCS CONCEPTS

• **Networks** → **Cloud computing**; • **Theory of computation** → **Rounding techniques**.

KEYWORDS

Service Chain Embedding, Data Sharing, Randomized Rounding.

This work was supported in part by the National Science Foundation under Grants CNS 1815676 and 1619129, the Army Research Office under Agreement Number W911NF18-10-378, and the Office of Naval Research under Grant N00014-19-1-2566.

ACM Reference Format:
Konstantinos Poularakis, Jaime Llorca, Antonia M. Tulino, and Leandros Tassiulas. 2020. Approximation Algorithms for Data-Intensive Service Chain Embedding. In *The Twenty-first ACM International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (Mobihoc '20), October 11–14, 2020, Boston, MA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397166.3409149

1 INTRODUCTION

1.1 Motivation

Driven by advances in network virtualization and programmability, recent years have seen a paradigm shift in cloud computing, from centralized towards distributed cloud architectures such as Fog [1] and Mobile Edge Computing (MEC) [2]. Harvesting cloud resources distributed throughout the network core and edge in proximity to end-users (e.g., at switches and base stations) provides a clear advantage in performing tasks with stringent latency requirements that are hard to satisfy from centralized cloud locations.

These trends enable the realization of interesting new service models such as *Service Chaining (SC)* [3]. In a nutshell, a service chain describes a sequence of service functions that a flow needs to pass through in a particular order so as to get a complete end-to-end service. For example, as depicted in Figure 1, a service chain could define that a flow is first routed from its source to a firewall for security, next through a deep packet inspection (DPI) and a load balancer for traffic optimization, and only then is delivered to its destination.

A key aspect driving both performance and efficiency of a SC is how to instantiate it onto the substrate network. This necessitates the placement of service functions in the network, as well as the routing (or steering) of flows through the appropriate function instances. This problem is further complicated by the *multi-dimensional resource requirements* of the services. Many services today require not only computation and communication resources for executing processing tasks and delivering associated outputs to the end users, but also non-trivial amounts of data that should be stored in advance in the network and be available for the service functions to access in an on demand manner. Examples of such data-intensive services are Face Recognition and Augmented/Virtual Reality (AR/VR) that require the availability of databases of images and visual recognition models, possibly of several gigabytes each, in order to run classification and recognition functions [4].

Storage differs from computation and communication resources in that a stored data object can be *shared* to satisfy service requests

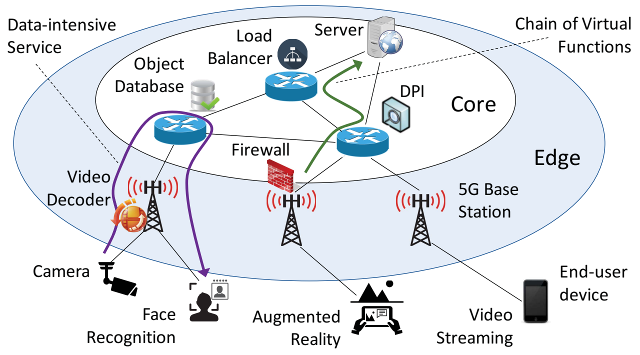


Figure 1: An illustrative distributed cloud network. Requests for data-intensive services are satisfied by steering traffic through chains of service functions instantiated at core and edge cloud nodes.

with the same or overlapping data requirements (e.g., the same object database or video frames). In contrast, computation and communication resources are typically allocated in a *dedicated* manner such that each service request takes up its own portion of resources for exclusive use and the total resource consumption at a cloud node or a network link is the sum of resource allocations.

Major network operators have made large investments in recent years in building both centralized and distributed data centers and developing their own software solutions to facilitate the large-scale storage and sharing of data among services. For example, the *network data layer solution* [5] enables the decoupling between storage and computation functions by providing data storage separately from a distributed shared database. Such decoupling allows the independent scaling of storage and computation functions and brings increased operation flexibility.

Despite the above industry efforts, a fundamental algorithmic methodology for optimally instantiating chains of data-intensive services into the network is still missing. Existing algorithms based on *SC Embedding (SCE)* (e.g., [6], [7], [8]) have been effective in addressing this problem by provisioning computation together with appropriate communication resources to minimize network costs or maximize accepted service requests. However, the shareable nature of the stored data breaks the suitability of these algorithms. A few recent works studied the impact of data sharing on SCE (e.g., see [9], [10], [11] and the discussion of related works in Section 2). However, these works focused on services with single (not chained) functions and required that the data object is colocated with the function using it (hence sharing of data is constrained within the boundaries of the same cloud node). Also, these works considered specific (not arbitrary) network topologies representing the last-mile (single-hop) connectivity between cloud nodes at base stations and mobile users.

1.2 Methodology & Contributions

In this paper, we follow a systematic methodology to optimize the deployment of data-intensive service chains, summarized as follows:

(1) We employ general network and service models to represent the deployment of data-intensive service chains in distributed cloud systems with three-dimensional (storage, computation and communication) resource constraints. SCs are modeled by Directed Acyclic Graphs (DAGs) while networks of arbitrary topology and with multi-hop and multi-path connectivity are considered.
(2) We formulate the Data-intensive Service Chain Embedding (DSCE) problem mathematically as an integer programming problem with the goal of minimizing storage, computation, and communication resource costs subject to resource capacity, service chaining, and data sharing constraints. Our formulation differs from those in classic SCE literature in that it contains not only variables for the embedding of service functions and flows onto the substrate network, but also data placement variables that capture the sharing of data objects between service functions and end user requests.
(3) Using a randomized rounding technique that exploits a novel data-aware linear programming decomposition procedure, we develop a multi-criteria algorithm that provably achieves approximation guarantees while violating the resource capacities in a bounded way. To the best of our knowledge, this is the first approximation algorithm for this problem.
(4) We perform evaluations to show the efficiency of the proposed algorithm. We find that, in many practical scenarios, our algorithm achieves near optimal cost that is up to 27.8% better than the case where storage is treated as a dedicated resource and thus sharing of data among services is not permitted. In most cases, there is at least one embedding extracted by our algorithm that has minimal (less than 3%) capacity violation while the violation factors vanish as the available capacities increase.

The rest of the paper is organized as follows. Section 2 discusses related works, whereas Section 3 presents the system model and formulates the data-intensive service chain embedding problem. We present a solution algorithm with approximation guarantees in Section 4. Finally, we present the evaluation results in Section 5 and conclude our work in Section 6.

2 RELATED WORK

Most of the previous related works focused on services without intensive data requirements (where the challenge is how to allocate computation and communication resources that are dedicated/non-shareable) or without chaining constraints (where a single function constitutes a service). A survey of related works can be found in [12] whereas a rough categorization based on the assumptions made by these works is shown in Figure 2 and discussed below. We note that our DSCE problem generalizes all this body of works.

2.1 Services without intensive data requirements

Services such as firewalls, network address translators, load balancers and video encoders are increasingly deployed in the form of virtual functions by network operators. Services of this family typically *do not impose intensive data requirements* and therefore their optimization depends on the allocation of computation and

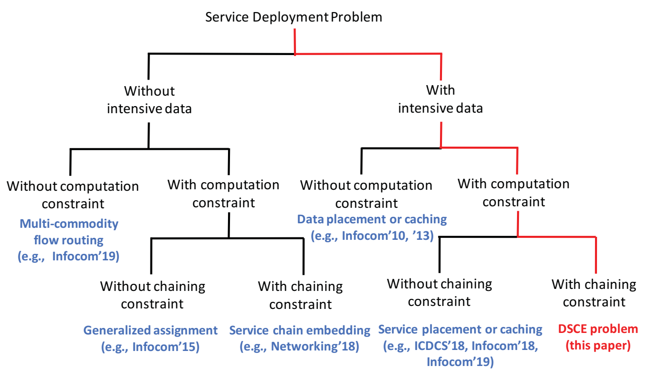

Service Deployment Problem
Without intensive data
With intensive data
Without computation constraint
With computation constraint
Without computation constraint
With computation constraint
Multi-commodity flow routing (e.g., Infocom'19)
Data placement or caching (e.g., Infocom'10, '13)
Without chaining constraint
With chaining constraint
Without chaining constraint
With chaining constraint
Generalized assignment (e.g., Infocom'15)
Service chain embedding (e.g., Networking'18)
Service placement or caching (e.g., ICDCS'18, Infocom'18, Infocom'19)
DSCE problem (this paper)

Figure 2: Classification of the related work for the service deployment problem.

communication rather than storage resources. In the extreme case that services do not consume even computation (or when computation capacity is not the bottleneck), then the only requirement is to route traffic flows over the capacitated links among functions in a network, which can be casted as a variant of the *multi-commodity flow routing problem* [13]. For the more practical case that functions require to execute some computation task, the work in [14] formulated the computation and communication allocation problem as a *generalized assignment problem* for which approximation algorithms are known. Subsequent works in [15] and [16] proposed integer linear programming formulations for the objectives of minimizing cost or maximizing accepted requests, and solved the problem using bargaining and conformal mapping theory, respectively.

The problem is further complicated, however, if one considers order constraints in the execution of the functions known as chaining constraints. Chained service functions can be modeled as service graphs to be *embedded* into a substrate network and various algorithms have been proposed for this purpose using techniques such as randomized rounding [6], relaxation and successive convex approximation [7], and shortest path routing on a multi-layer graph [8]. Alternative modeling and solution approaches are also known based on mixed integer linear programming [17], [18], column generation [19], sampling-based Markov approximation [20] and queuing models [21]. However, the above works did not consider functions with intensive data requirements or treated data storage as another dedicated non-shareable resource to allocate, just like computation and communication.

2.2 Services with intensive data requirements

Modern services such as Face Recognition and Augmented Reality require access to non-trivial amounts of data (e.g., data base of images, visual models, etc) which should be stored in advance at the cloud nodes and be available for the functions to share in an on demand manner. This can be formulated as a *data placement problem* [22] also referred as *caching problem* [23], [24] and is among the most well-investigated problems in networking literature. However, the data placement/caching problem is much simpler than the service deployment problem since it does not deal with the allocation of computation resources, the execution of functions and the accompanying chaining constraints. Still, this problem is NP-Hard due to the combinatorial nature of the data placement decisions.

Recently, there have been some works that generalize the data placement problem by adding computation allocation variables and computation constraints to satisfy requests for data-intensive services [9], [10], [11]. This generalized problem is referred as the *service placement problem* and several approximation algorithms for jointly allocating storage, computation, and communication resources have been proposed. While these works considered the shareable nature of the stored data, they assumed that each service request is for a single function rather than a chain of functions, and that data objects must be colocated with the services using them at the same cloud location. Besides, these works focused on two-tier (single-hop) network topologies.

To the best of our knowledge, our work is the first to consider all these aspects together (shareable nature of stored data, decoupling of data objects from computation functions, service chaining constraints, and arbitrary multi-hop multi-path network topologies) and generalizes the previous works in this area.

3 MODEL AND PROBLEM FORMULATION

In this section, we formally introduce the network and service models that will be later used to optimize the deployment of data-intensive service chains in the distributed cloud network.

3.1 Network Model

We model the distributed cloud network as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V}$ vertices and $\mathcal{E}$ edges representing the set of network nodes and links, respectively. A node in $\mathcal{V}$ may represent an end-user device, an access point or a cloud node inside the core or edge network. Each node $u \in \mathcal{V}$ is characterized by its computation resources (e.g., processor, microprocessor) and storage resources (e.g., hard disk, flash memory). We denote by C_u and R_u the computation and storage capacities at node $u \in \mathcal{V}$, respectively. The cost of allocating one unit of processing and storage resource at node u is c_u and r_u, respectively. Nodes are interconnected via wireless or wireline links, each characterized by its transmission capacity and unit cost. Link (u, v) has transmission capacity B_{uv} and the cost per bandwidth resource unit is b_{uv}.

3.2 Service Model

A generic service $\phi \in \Phi$ can be described by a directed acyclic graph (DAG) $\mathcal{G}^{\phi} = (\mathcal{V}^{\phi}, \mathcal{E}^{\phi})$ where vertices represent service functions and edges represent streams of information exiting one function and entering another. An example of a service graph is shown in Figure 3. There exist the following four types of vertices in the service graph $\mathcal{G}^{\phi}$:

(1) *Source or production functions* ($\mathcal{V}^{\phi,s}$ set) are vertices with no incoming edges of the service graph. They represent sensors that generate real-time data streams (e.g., video cameras, IoT sensors) associated with specific locations in the physical network (e.g., user devices, nearby access points).
(2) *Storage functions* ($\mathcal{V}^{\phi,r}$ set) are also vertices with no incoming edges of the service graph. They represent the storage/caching of pre-generated static data (e.g., pre-coded

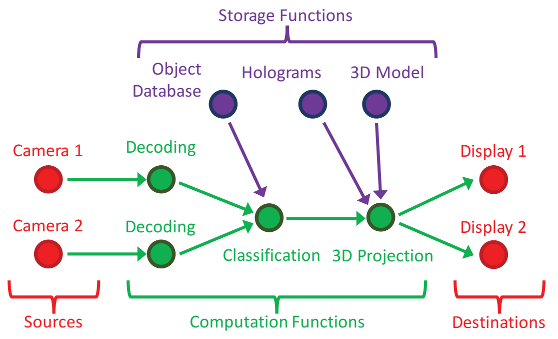


Figure 3: Service graph example; flows from two video cameras go through decoding, classification, and 3D projection functions before being displayed at two consumer devices. Storage functions provide data as input to computation functions.

video). There is one storage function for each computation function that requires access to a given data object. Hence, if multiple computation functions require access to the same data object, we still create multiple storage functions.

(3) *Computation functions* ($\mathcal{V}^{\phi,c}$ set) are internal vertices of the service graph. They take as input source, storage, or other computation functions, and pass their output to other computation or destination functions.

(4) *Destination or consumption functions* ($\mathcal{V}^{\phi,d}$ set) are vertices with no outgoing edges of the service graph. They represent end devices (e.g., video displays), consuming output information streams and are associated with specific locations in the physical network, just like the source functions.

We define the storage and computation requirements of the service functions (vertices of the service graph) as follows. A storage function $i \in \mathcal{V}^{\phi,r}$ has zero computation requirement but requires to store a data object (e.g., video, object database, holograms for an AR service, etc.) which consumes storage resource. We denote by $o(i)$ and $r^{o(i)} > 0$ the respective data object and the required storage resource, where the latter clearly depends on the size of the data object. A computation function $i \in \mathcal{V}^{\phi,c}$ has zero storage requirement but requires computation resource $c^i > 0$ for task execution. The rest of vertices (source and destination functions) have zero storage and computation requirements.

Note that differently from existing works on single-function placement with storage requirements (e.g., [9–11]), where a copy of the data needs to be colocated with each function requiring access to that data, our model decouples data from computation, allowing the possibility of multiple functions sharing access to a common copy of the data. We emphasize that multiple storage functions belonging to the same or different service graphs may require to store the same data object. For example, it may happen that $o(i) = o(i')$ for two storage functions $i \in \mathcal{V}^{\phi,r}$ and $i' \in \mathcal{V}^{\phi',r}$ where $\phi = \phi'$ or $\phi \neq \phi'$. In this case, the data object can be *shared* by the storage functions reducing the overall storage consumption. This is a major difference compared to computation and bandwidth resources which are non-sharable meaning that each individual function takes up its own *dedicated* portion of resources for exclusive use. We denote by $\mathcal{R}(o)$ the set of all storage functions across all service graphs $\phi \in \Phi$ requiring object o.

Next, we define the bandwidth requirements of the service streams (edges of the service graph). We denote by b^{ij} the bandwidth requirement of stream $(i, j) \in \mathcal{E}^{\phi}$. For streams generated by a source or computation function i, b^{ij} depends on the volume of generated data (e.g., video bitrate). For streams coming out of a storage function i, b^{ij} depends on the size of the stored data and the rate at which function j accesses the data.

To facilitate presentation, we denote by $\mathcal{G}^{\Phi} = (\mathcal{V}^{\Phi}, \mathcal{E}^{\Phi})$ the union of all service graphs, where $\mathcal{V}^{\Phi} = \cup_{\phi} \mathcal{V}^{\phi}$ and $\mathcal{E}^{\Phi} = \cup_{\phi} \mathcal{E}^{\phi}$. Similarly, $\mathcal{V}^{\Phi,s}$, $\mathcal{V}^{\Phi,r}$, $\mathcal{V}^{\Phi,c}$ and $\mathcal{V}^{\Phi,d}$ denote the unions of all source, storage, computation and destination functions for all service graphs. We also denote by $\mathcal{S}(u)$ and $\mathcal{D}(u)$ the sets of all source and destination functions associated with a particular node u. Finally, we denote by $O = \cup_i o(i)$ the union of all data objects.

3.3 Mathematical Formulation

The network operator needs to decide how to embed (or map) the vertices and edges of the collection of service graphs to the nodes and links of the physical substrate network. To model the embedding decisions, we introduce two sets of optimization variables: (i) $x_u^i \in \{0, 1\}$ which indicates whether vertex i is mapped to physical node u ($x_u^i = 1$) or not ($x_u^i = 0$), and (ii) $y_{uv}^{ij} \in \{0, 1\}$ which indicates whether edge (i, j) is mapped to a path that uses physical link (u, v) ($y_{uv}^{ij} = 1$) or not ($y_{uv}^{ij} = 0$).

Differently from traditional service embedding problem formulations (e.g., [6]), in our case, the network operator needs to also decide the data placement decisions: $z_u^o \in \{0, 1\}$ which indicates whether data object o is stored at physical node u ($z_u^o = 1$) or not ($z_u^o = 0$). The respective vectors of variables are the following:

$$\boldsymbol{x} = (x_u^i \in \{0, 1\} \;: u \in \mathcal{V}, i \in \mathcal{V}^{\Phi}) \tag{1}$$

$$\boldsymbol{y} = (y_{uv}^{ij} \in \{0, 1\} \;: (u, v) \in \mathcal{E}, (i, j) \in \mathcal{E}^{\Phi}) \tag{2}$$

$$\boldsymbol{z} = (z_u^o \in \{0, 1\} \;: u \in \mathcal{V}, o \in O) \tag{3}$$

The above decisions need to satisfy several constraints. First, we need to make sure that each source and destination function is mapped to its associated node in the physical network:

$$x_u^i = 1, \; \forall u \in \mathcal{V}, i \in \mathcal{S}(u) \cup \mathcal{D}(u) \tag{4}$$

Second, we need to map each storage and computation function to exactly one node in the physical network:

$$\sum_{u \in \mathcal{V}} x_u^i = 1, \; \forall i \in \mathcal{V}^{\Phi,r} \cup \mathcal{V}^{\Phi,c} \tag{5}$$

Third, we need to ensure that each edge $(i, j) \in \mathcal{E}^{\Phi}$ is mapped to a path in the physical network that starts at the location of function i and ends at the location of function j, hence respecting service chaining requirements:

$$\sum_{(u,v) \in \mathcal{E}} y_{uv}^{ij} - \sum_{(v,u) \in \mathcal{E}} y_{vu}^{ij} = x_u^i - x_u^j, \; \forall u \in \mathcal{V}, (i, j) \in \mathcal{E}^{\Phi} \tag{6}$$

Fourth, the total computation load of services running on physical node u must not exceed its computation capacity:

$$\sum_{i \in \mathcal{V}^{\Phi,c}} x_u^i c^i \leq x_u \leq C_u, \ \forall u \in \mathcal{V} \tag{7}$$

where x_u is an auxiliary variable whose value is equal to the computation load of node u in the optimal solution of the problem. Fifth, the total bandwidth load of services transported over physical link (u, v) must not exceed its bandwidth capacity:

$$\sum_{i,j \in \mathcal{E}^{\Phi}} y_{uv}^{ij} b^{ij} \leq y_{uv} \leq B_{uv}, \ \forall (u,v) \in \mathcal{E} \tag{8}$$

where y_{uv} is an auxiliary variable whose value is equal to the bandwidth load of link (u, v) in the optimal solution. To satisfy the data requirements of services, we also need to satisfy the following two sets of constraints:

$$x_u^i \leq z_u^{o(i)}, \ \forall u \in \mathcal{V}, i \in \mathcal{V}^{\Phi,r} \tag{9}$$

$$\sum_{o \in \mathcal{O}} z_u^o r^o \leq z_u \leq R_u, \ \forall u \in \mathcal{V} \tag{10}$$

The first of the two constraints ensures that a storage function i can be mapped to a node u ($x_u^i = 1$) only if the latter has stored the required data object $o(i)$. The second set of constraints ensures that the total storage capacity of a node u is not exceeded, where z_u is an auxiliary variable whose value is equal to the storage load of node u in the optimal solution. It is important to note that the storage resource load is computed using the data placement variables $z_u^o, o \in \mathcal{O}$, and not the storage function variables $x_u^i, i \in \mathcal{V}^{\Phi,r}$, hence allowing the sharing of storage resources between multiple functions storing the same data object.

The goal of the network operator is to make the embedding $(\boldsymbol{x},\boldsymbol{y})$ and data placement (z) decisions that minimize the total resource consumption costs:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}} \quad & \textstyle\sum_{u \in \mathcal{V}} (x_u c_u + z_u r_u) + \sum_{(u,v) \in \mathcal{E}} y_{uv} b_{uv} \\ \text{s.t.} \quad & \text{constraints: } (1) - (10) \end{aligned} \tag{11}$$

We refer to the above as the *Data-Intensive Service Chain Embedding* (*DSCE*) problem. It is not difficult to show that the DSCE problem is NP-Hard since it generalizes the data placement problems in [22], [23] and [24] by augmenting additional variables and constraints into the problem formulation. Therefore, it is unlikely to find optimal solutions and the use of approximation algorithms is justified. In the next section, we present such an algorithm and formally prove its approximation guarantees.

4 APPROXIMATION ALGORITHM

In this section, we present one of the main contribution of this paper; an approximation algorithm for the DSCE problem. Our algorithm, termed DSCE-RR, leverages a Randomized Rounding technique that chooses an embedding of the service collection $\mathcal{G}^{\Phi}$ extracted from the linear programming (LP) relaxation of the DSCE problem following a novel data-aware decomposition procedure.

4.1 Algorithm Description

The DSCE-RR algorithm is described in Algorithm 1. DSCE-RR iteratively extracts service embeddings from the LP solution $(\bar{x}, \bar{y}, \bar{z})$ (line 2) by mapping each vertex and edge in the service collection $\mathcal{G}^{\Phi}$ onto the substrate network $\mathcal{G}$. It uses Algorithm 2 to map the set of source functions (line 7), Algorithm 3 to map the set of storage functions (line 8), and Algorithm 4 to map the sets of computation functions, destination functions, and service edges (line 9).

Of particular relevance is the procedure that maps the storage functions in Algorithm 3. Note how storage functions are mapped by first finding a physical node u with positive fractional data placement value $\bar{z}_u^o$ for each unique object $o \in \mathcal{O}$ (line 6), and then mapping to that location all storage functions in $\mathcal{R}(o)$ with positive fractional placement value x_u^i (lines 8-12). Importantly, *this procedure favors the extraction of service embeddings from the LP solution that colocate storage functions associated with the same object*, which in turn leads to more sharing of data among services and saves storage resources.

Once source and storage functions are mapped via Algorithm 2 and Algorithm 3, and the set Q containing currently mapped functions is updated, the DSCE-RR algorithm calls Algorithm 4 for mapping the service edges (representing data streams) and the remaining computation and destination functions. Algorithm 4 visits every single edge in the service graph $\mathcal{G}^{\Phi}$ and maps it onto a path in the physical network $\mathcal{G}$, mapping also the remaining computation and destination functions. Note how in the case that we have nodes with multiple incoming edges in $\mathcal{G}^{\Phi}$, Algorithm 4 makes sure those

Algorithm 1: DSCE-RR Algorithm

```
1  Input: Network graph $\mathcal{G}$, service graph $\mathcal{G}^{\Phi} = \cup_{\phi} \mathcal{G}^{\phi}$
2  Solve the linear relaxation of the DSCE problem to obtain $(\bar{x}, \bar{y}, \bar{z})$
3  Set $\mathcal{M} = \emptyset, \gamma = 1, k = 1$
4  while $\gamma > 0$ do
5      Set $m_k = (m_k^V, m_k^E) = (\emptyset, \emptyset)$
6      Set $Q = \emptyset$
7      Compute mapping of source nodes according to Algorithm 2
8      Compute mapping of storage nodes according to Algorithm 3
9      Compute mapping of computation nodes, destination nodes and service
         edges according to Algorithm 4
10     Set $\mathcal{W}_k = \{\bar{x}_u^i \mid i \in \mathcal{V}^{\Phi}, u = m_k^V(i)\}$
               $\cup \{\bar{y}_{uv}^{ij} \mid (i,j) \in \mathcal{E}^{\Phi}, (u,v) \in m_k^E(i,j)\}$
11     Set $p_k = \min \mathcal{W}_k$
12     Set $w = w - p_k, \forall w \in \mathcal{W}_k$
13     Set $\gamma = \gamma - p_k$
14     Set $\mathcal{M} = \mathcal{M} \cup (m_k, p_k)$
15     Set $k = k + 1$
   end
16 Choose embedding $m_k$ with probability $p_k$
17 Set $\widehat{x}_u^i = 1, \forall i \in \mathcal{V}^{\Phi}, u \in m_k^V(i)$
18 Set $\widehat{y}_{uv}^{ij} = 1, \forall (i,j) \in \mathcal{E}^{\Phi}, u \in m_k^E(i,j)$
19 Set $\widehat{z}_u^o = \max_{i \in \mathcal{R}(o)} \widehat{x}_u^i, \forall o \in \mathcal{O}, u \in \mathcal{V}$
20 Output: $\widehat{x}, \widehat{y}, \widehat{z}$
```

Algorithm 2: Source Node Mapping

```
1  Input: $\bar{x}, m_k^V, Q$
2  for $i \in \mathcal{V}^{\Phi,s}$ do
3      Set $Q = Q \cup \{i\}$
4      Find the node $u \in \mathcal{V}$ with $\bar{x}_u^i = 1$
5      Set $m_k^V(i) = u$
   end
6  Output: $m_k^V, Q$
```

Algorithm 3: Storage Node Mapping

```
1  Input: $\bar{x}, \bar{z}, m_k^V, Q$
2  for $o \in O$ do
3      Set $\mathcal{X} = \mathcal{R}(o)$
4      Set $\mathcal{Y} = \mathcal{V}$
5      while $|\mathcal{X}| > 0$ do
6          Pick a node $u \in \mathcal{Y}$ with $\bar{z}_u^o > 0$
7          Set $\mathcal{Y} = \mathcal{Y} \backslash \{u\}$
8          for $i \in \mathcal{X}$ do
9              if $\bar{x}_u^i > 0$ then
10                 Set $m_k^V(i) = u$
11                 Set $Q = Q \cup \{i\}$
12                 Set $\mathcal{X} = \mathcal{X} \backslash \{i\}$
               end
           end
       end
   end
13 Output: $m_k^V, Q$
```

Algorithm 4: Computation Node and Link Mapping

```
1  Input: $\bar{x}, \bar{y}, m_k^V, m_k^E, Q$
2  while $|Q| > 0$ do
3      Pick a node $i \in Q$ and remove it from set $Q$
4      for $(i, j) \in \mathcal{E}^\Phi$ do
5          if $m_k^V(j) \neq \emptyset$ then
6              Find a path $\mathcal{P}$ connecting $u = m_k^V(i)$ to $v = m_k^V(j)$, such
               that $\bar{y}_{ab}^{ij} > 0 \; \forall (a, b) \in \mathcal{P}$
7              Set $m_k^E(i, j) = \mathcal{P}$
           end
8          else
9              Find a path $\mathcal{P}$ connecting $u = m_k^V(i)$ to a $v \in \mathcal{V}$ with
               $\bar{x}_v^j > 0$, such that $\bar{y}_{ab}^{ij} > 0 \; \forall (a, b) \in \mathcal{P}$
10             Set $m_k^V(j) = v$
11             Set $m_k^E(i, j) = \mathcal{P}$
           end
12         Set $Q = Q \cup \{j\}$
       end
   end
13 Output: $m_k^V, m_k^E, Q$
```

computation functions are only mapped once. Note also how the procedure in Algorithm 4 terminates when all destination functions are mapped to their corresponding physical hosts.

After completing the mapping procedures in lines 7-9, DSCE-RR computes the probability p_k associated with the extracted embedding m_k as the minimum value among the LP variables associated with such embedding (lines 10-11), and subtracts it from all such variables (line 12), completing the extraction of embedding m_k. Note how the parameter γ that controls the termination of the decomposition procedure is also reduced by p_k after the extraction of embedding m_k (line 13). At the end of the decomposition procedure (when $\gamma = 0$), embedding m_k is chosen with probability p_k (lines 16). In the end (lines 17-19), the associated embedding and placement variables are computed.

4.2 Performance analysis

To facilitate the analysis of DSCE-RR algorithm, we introduce the following definition.

Definition 1. *A valid embedding $\mathcal{M} = (m^V, m^E)$ of service collection $\mathcal{G}^\Phi$ on network $\mathcal{G}$ consists of a node mapping $m^V : \mathcal{V}^\Phi \rightarrow \mathcal{V}$ and a link mapping $m^E : \mathcal{E}^\Phi \rightarrow \mathcal{E}$, such that:*

- *Each source function $i \in \mathcal{V}^{\Phi,s}$ is mapped to its corresponding host $\{u \in \mathcal{V} | i \in \mathcal{S}(u)\}$.*
- *Each destination function $i \in \mathcal{V}^{\Phi,d}$ is mapped to its corresponding host $\{u \in \mathcal{V} | i \in \mathcal{D}(u)\}$.*
- *Each storage function $i \in \mathcal{V}^{\Phi,r}$ is mapped to one physical node in $\mathcal{V}$ that contains a copy of data object $o(i) \in O$.*
- *Each computation function $i \in \mathcal{V}^{\Phi,c}$ is mapped to one physical node in $\mathcal{V}$.*
- *Each edge $(i, j) \in \mathcal{E}^\Phi$ is mapped to one path starting at node $m^V(i)$ and ending at node $m^V(j)$.*

We then show the following two lemmas.

Lemma 1. *For a given network graph $\mathcal{G}$ and service graph $\mathcal{G}^\Phi$, Algorithm 1 decomposes a solution $(\bar{x}, \bar{y}, \bar{z})$ to the LP relaxation of the DSCE problem into a convex combination of valid embeddings $\mathcal{M} = \{\mathcal{M}_k\}$ with $\mathcal{M}_k = \{m_k, p_k\}$, such that $\sum_k p_k = 1$.*

Proof. We first show the validity of the embeddings obtained by the DSCE-RR algorithm. To this end, we show that for each embedding, Algorithm 1 ends up mapping every node and link in $\mathcal{G}^\Phi$, and that each node and link mapping is valid. It is immediate to see that the mapping of the source nodes is valid since source i is mapped to the only physical node u where $\bar{x}_u^i = 1$ (Algorithm 2, line 4), i.e., the node where the placement of i was initialized to via constraint (4). Let's now focus on the storage nodes. Note that each storage function $i \in \mathcal{V}^{\Phi,r}$ is associated with a unique data object $o(i)$. For each $o \in O$, Algorithm 3 updates the set of current unmapped storage functions $\mathcal{X}$ associated with a given object every time a new function is mapped, and it runs until the set is empty. Hence, each storage function is mapped exactly once. In addition, Line 6 assures that the node u to which storage function i is mapped has positive placement value $\bar{z}_u^o$. The mapping of computation nodes is valid since 1) for a given edge $(i, j) \in \mathcal{E}^\Phi$, Algorithm 4 can only map function j after function i has been mapped, 2) function j can only be mapped to physical node v if there is a path from the location of already mapped function i to v where each edge (a, b) on the path has positive $\bar{y}_{ab}^{ij}$ value (lines 6 and 9 in Algorithm 4), and 3) line 5 assures that if function j is visited more than once, it is only mapped once. Link mappings are valid since (i, j) is always mapped to a path starting at the already mapped location of i and ending at either the already mapped location of j (line 6) or a valid location of j (line 9), via a path whose edges have positive $\bar{y}_{ab}^{ij}$ values. Finally, since each service node and link will be eventually mapped, the overall embedding is valid. We now show that the decomposition of valid embeddings is complete. Note that since γ starts with value 1, its value is reduced by p_k at each iteration, and the procedure continues as long as $\gamma > 0$, then $\sum_k p_k$ cannot be smaller than 1. On the other hand, since at each iteration at least one variable's value is set to 0, then $\sum_k p_k$ cannot be larger than 1. Hence, Algorithm 1 obtains a decomposition $\mathcal{M} = \{\mathcal{M}_k\}$ for which $\sum_k p_k = 1$. □

Lemma 2. *Let $\widehat{\boldsymbol{x}}(k), \widehat{\boldsymbol{y}}(k), \widehat{\boldsymbol{z}}(k)$ denote the solution obtained by Algorithm 1 when the chosen embedding is $\mathcal{M}_k$. Then, the expected value of the solution obtained by Algorithm 1 is equal to the solution to the LP relaxation of DSCE, i.e.,*

$$\mathbb{E}[\widehat{\boldsymbol{x}}] = \sum_k p_k \widehat{\boldsymbol{x}}(k) = \bar{\boldsymbol{x}}$$
$$\mathbb{E}[\widehat{\boldsymbol{y}}] = \sum_k p_k \widehat{\boldsymbol{y}}(k) = \bar{\boldsymbol{y}}$$
$$\mathbb{E}[\widehat{z}] = \sum_k p_k \widehat{z}(k) = \bar{z}$$

Proof. We first show that the decomposition of Algorithm 1 is complete, in the sense that at the end of the decomposition procedure, the residual values of LP variables $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}, \bar{z})$ are all zero. To this end, note that $\sum_u \bar{x}_u^i, \forall i$, is initially equal to 1 by constraint (5), and that at each iteration such value is reduced by p_k (since exactly one node u will be chosen as the mapping of i). Given that $\sum_k p_k = 1$ from Lemma 1, it follows that at the end of the decomposition procedure, $\sum_u \bar{x}_u^i - \sum_k p_k = 0, \forall i$, and hence all $\bar{\boldsymbol{x}}$ residual values are zero. It is clear from constraints (6) and (10) that if all $\bar{\boldsymbol{x}}$ residual values are zero, then the residual values of $\bar{\boldsymbol{y}}$ and $\bar{z}$ must also be zero.

Now focus on a particular LP variable $\bar{x}_u^i$. At iteration k, its value is decreased by p_k if u is included as the mapping of i in embedding k, and by zero otherwise. Since the decomposition is complete and all residual values end up being zero, then $\sum_k p_k 1\{m_k^V(i) = u\} = \bar{x}_u^i$. Notice that the solution of Algorithm 1 for $\widehat{x}_u^i$ is a Bernoulli random variable with parameter $\sum_k p_k 1\{m_k^V(i) = u\}$. Hence, $\mathbb{E}[\widehat{x}_u^i] = \sum_k p_k 1\{m_k^V(i) = u\} = \bar{x}_u^i, \forall i, u$. A similar reasoning follows for the $\widehat{\boldsymbol{y}}$ and $\widehat{z}$ variables. □

Building upon the above lemmas, the following theorem follows.

Theorem 1. *Let J and $\widehat{J}$ denote the optimal objective value and the objective value obtained by Algorithm 1, respectively. Then,*

$$\mathbb{P}\left(\widehat{J} \geq \beta_J J\right) \leq \left(\frac{\epsilon}{\Omega(1)}\right)^{2\Gamma_J} \tag{12}$$
$$\mathbb{P}\left(\widehat{x}_u \geq \beta_c C_u\right) \leq \left(\frac{\epsilon}{|\mathcal{V}|}\right)^{2\Gamma_c} \tag{13}$$
$$\mathbb{P}\left(\widehat{y}_{uv} \geq \beta_b B_{uv}\right) \leq \left(\frac{\epsilon}{|\mathcal{E}|}\right)^{2\Gamma_b} \tag{14}$$
$$\mathbb{P}\left(\widehat{z}_u \geq \beta_r R_u\right) \leq \left(\frac{\epsilon}{|\mathcal{V}|}\right)^{2\Gamma_r} \tag{15}$$

where $\epsilon \in (0,1)$ and $\Gamma_{(\cdot)} \in \mathbb{R}^+$ are constants satisfying $2\Gamma_{(\cdot)} > 1$. The terms $\beta_J = 1+\sqrt{\Gamma_J \Delta_J \log(\Omega(1)/\epsilon)}$, $\beta_c = 1+\sqrt{\Gamma_c \Delta_c \log(|\mathcal{V}|/\epsilon)}$, $\beta_b = 1 + \sqrt{\Gamma_b \Delta_b \log(|\mathcal{E}|/\epsilon)}$, and $\beta_r = 1 + \sqrt{\Gamma_r \Delta_r \log(|\mathcal{V}|/\epsilon)}$ are factors of exceeding the optimal objective value and violating the resource capacities, where $\Delta_J = (J_{max}/\bar{J})^2$, $\Delta_c = (L_u^c/C_u)^2$, $\Delta_b = (L_{uv}^b/B_{uv})^2$, $\Delta_r = (L_u^r/R_u)^2$, with $J_{max} = \max_k \widehat{J}(k)$, $L_u^c = \max_k \widehat{x}_u(k)$, $L_{uv}^c = \max_k \widehat{y}_{uv}(k)$, $L_r^c = \max_k \widehat{z}_u(k)$ denoting the maximum objective value and resource loads over all extracted embeddings.

Proof. We first bound the probability that the computation load at a given node $\widehat{x}_u$ exceeds capacity C_u by a factor β_c. We have that:

$$\begin{aligned}
\mathbb{P}\left(\widehat{x}_u \geq \beta_c C_u\right) &\leq \mathbb{P}\left(\widehat{x}_u - \bar{x}_u \geq (\beta_c - 1)C_u\right) && (16)\\
&\leq \exp\left[-\frac{2((\beta_c-1)C_u)^2}{(L_u^c)^2}\right] && (17)\\
&= \exp\left[-\frac{2\Gamma_c \Delta_c \log(|\mathcal{V}|/\epsilon) C_u^2}{(L_u^c)^2}\right] && (18)\\
&= \exp\left[-2\Gamma_c \log(|\mathcal{V}|/\epsilon)\right] && (19)\\
&= \left(\frac{\epsilon}{|\mathcal{V}|}\right)^{2\Gamma_c} && (20)
\end{aligned}$$

Note that (16) follows from the fact that $\bar{x}_u \leq C_u$ due to (7), and that $\mathbb{E}[\widehat{x}_u] = \bar{x}_u$ due to Lemma 2; (17) is due to Hoeffding's lemma [25] applied for one bounded variable $\widehat{x}_u \in [0, L_u^c]$; (18) follows from setting $\beta_c - 1 = \sqrt{\Gamma_c \Delta_c \log(|\mathcal{V}|/\epsilon)}$ with $\epsilon \in (0,1)$, $\Gamma_c \in \mathbb{R}^+$ and $\Delta_c > 0$; and (19) follows from the definition of Δ_c.

Following an equivalent procedure and choosing $\beta_b = 1 + \sqrt{\Gamma_b \Delta_b \log(|\mathcal{E}|/\epsilon)}$ with $\Delta_b = (L_{uv}^b/B_{uv})^2$, and $\beta_r = 1+\sqrt{\Gamma_r \Delta_r \log(|\mathcal{V}|/\epsilon)}$ with $\Delta_r = (L_u^r/R_u)^2$, (14) and (15) follow.

Finally, in order to prove (12), letting $\bar{J}$ denote the objective function value of the optimal fractional solution, we have:

$$\begin{aligned}
\mathbb{P}\left(\widehat{J} \geq \beta_J J\right) &\leq \mathbb{P}\left(\widehat{J} \geq \beta_J \bar{J}\right) && (21)\\
&\leq \exp\left[-\frac{2((\beta_J - 1)\bar{J})^2}{(J_{max})^2}\right] && (22)\\
&= \exp\left[-\frac{2\Gamma_J \Delta_J \log(\Omega(1)/\epsilon)\bar{J}^2}{(J_{max})^2}\right] && (23)\\
&= \exp\left[-2\Gamma_J \log(\Omega(1)/\epsilon)\right] && (24)\\
&= \left(\frac{\epsilon}{\Omega(1)}\right)^{2\Gamma_J} && (25)
\end{aligned}$$

□

We now introduce the following definition and state the approximation guarantees of the DSCE-RR algorithm.

Definition 2. *A multi-criteria $(\beta_J, \beta_c, \beta_b, \beta_r)$-approximation for the DSCE problem is a solution where the objective value exceeds that of the optimal solution by at most a factor of β_J, while the computation, bandwidth, and storage capacity constraints are violated by at most a factor of β_c, β_b, and β_r, respectively.*

Theorem 2. *As the size of $\mathcal{G}$ and $\mathcal{G}^\Phi$ grows, the DSCE-RR algorithm returns with high probability a multi-criteria $(\beta_J, \beta_c, \beta_b, \beta_r)$-approximation for the DSCE problem, with $(\beta_c, \beta_b, \beta_r)$ as given in Theorem 1, $\beta_J = 1 + \sqrt{\Gamma_J \Delta_J \log(\Theta(|\mathcal{V}|)/\epsilon)}$, $\Gamma_c, \Gamma_b, \Gamma_r > 1$ and $\Gamma_J > 0.5$.*

Proof. Using the union bound and Theorem 1, we have that the probability that any of the computation and storage capacities of the $|\mathcal{V}|$ nodes and the bandwidth capacities of the $|\mathcal{E}|$ links are violated by a factor β_c, β_r, and β_b, respectively, and the objective value $\widehat{J}$ exceeds the optimal objective value J by a factor β_J, is upper bounded by $|\mathcal{V}|\left(\frac{\epsilon}{|\mathcal{V}|}\right)^{2\Gamma_c} + |\mathcal{V}|\left(\frac{\epsilon}{|\mathcal{V}|}\right)^{2\Gamma_r} + |\mathcal{E}|\left(\frac{\epsilon}{|\mathcal{E}|}\right)^{2\Gamma_b} + \left(\frac{\epsilon}{\Theta(|\mathcal{V}|)}\right)^{2\Gamma_J}$, where we have replaced $\Omega(1)$ with $\Theta(|\mathcal{V}|)$ in the last term. This probability is equal to $\epsilon\left(\frac{\epsilon}{|\mathcal{V}|}\right)^{2\Gamma_c-1}+\epsilon\left(\frac{\epsilon}{|\mathcal{V}|}\right)^{2\Gamma_r-1}+\epsilon\left(\frac{\epsilon}{|\mathcal{E}|}\right)^{2\Gamma_b-1}+\left(\frac{\epsilon}{\Theta(|\mathcal{V}|)}\right)^{2\Gamma_J}$, which goes to zero as $|\mathcal{V}|$ and $|\mathcal{E}|$ increase for $\Gamma_c, \Gamma_r, \Gamma_b > 1$ and $\Gamma_J > 0.5$, from which Theorem 2 follows. □

Note that Algorithm 1 only considers one rounding try (see line 16). To improve the performance, we can consider multiple rounding tries and then pick the one with the minimum violations. This way, we can show the following high probability bound without requiring that the size of $\mathcal{G}$ and $\mathcal{G}^\Phi$ grows.

Theorem 3. *Given fixed-size $\mathcal{G}$ and $\mathcal{G}^\Phi$, the DSCE-RR algorithm returns, after n rounding tries, with probability $1-(\widetilde{\epsilon})^n$, a multi-criteria $(\beta_J, \beta_c, \beta_b, \beta_r)$-approximation for the DSCE problem, with $(\beta_c, \beta_b, \beta_r)$ as given in Theorem 1, $\beta_J = 1+\sqrt{\Gamma_J \Delta_J \log(1/\epsilon)}$, $\Gamma_{(\cdot)} > 0.5$, $\epsilon = \widetilde{\epsilon}/4$ and $\widetilde{\epsilon} \in (0,1)$.*

Proof. Following considerations analogous to the ones in the proof of Theorem 2, the statement in Theorem 3 readily follows. The term $\epsilon = \widetilde{\epsilon}/4$ is due to the application of union bound and the fact that there are 4 types of violation factors we seek to bound. □

5 EVALUATION RESULTS

In this section, we perform evaluations to show the efficiency of the proposed DSCE-RR algorithm. We find that, in practical scenarios, DSCE-RR achieves near optimal cost that is up to 27.8% better than the case where storage is treated as a dedicated resource and thus sharing of data among services is not permitted. In most scenarios, there is at least one embedding extracted by DSCE-RR that has minimal (less than 3%) capacity violation while the violation factors vanish as the available capacities increase.

5.1 Evaluation Setup

We consider a similar setup as in [18], depicted in Figure 4. Cloud resources (storage and computation) are distributed across nodes of different tiers in a network; a head office (HO) node at the higher tier that represents a centralized data center, intermediate office (IO) and end-office (EO) nodes, and base station (BS) nodes associated with mobile end-users at the bottom tier. We construct service chains with the structure shown on the right of the figure. Each chain contains a source and a destination vertex associated with specific locations in the cloud network as well as storage and computation functions, the location of which can be optimized. This structure can capture various data-intensive services such as Augmented Reality (AR).

Based on the above structure, we construct $\Phi = 100$ AR service chains that differ from one another in the location of source and destination functions and the requirements of storage and computation functions. Specifically, we locate randomly the source and destination of each service chain at one of the four BSs or the HO node representing the demand of mobile users associated with the BSs or the demand coming from other networks through the HO. The data object required by each storage function is drawn from a library of $|\mathcal{O}| = 100$ data objects. For each function, the object is picked randomly based on the Zipf probability distribution with slope value 1 which is a common assumption for several types of

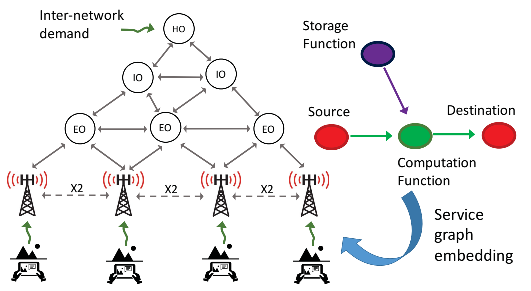


Figure 4: Distributed cloud network of $|\mathcal{V}| = 10$ nodes and $|\mathcal{E}| = 2 \times 18 = 36$ directed links (left) and service graph of $|\mathcal{V}^\phi| = 4$ vertices and $|\mathcal{E}^\phi| = 3$ edges (right).

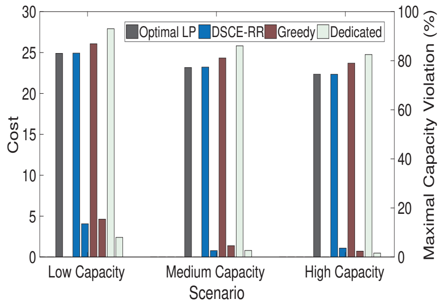
services [11]. This allows overlapping data requirements among services and therefore couples their embedding decisions.

We set the size of each data object randomly within the interval [1, 20] GBs. The bandwidth required for a source (storage function) to pass input to a computation function is set randomly (proportional to the object size) within [1, 10] Mbps. The output stream of a computation function is passed to the destination node at the aggregate data rate of the input streams coming from the source and storage functions. The computation requirement is set within [0.2,4] GHz assuming 200 cycles per bit of input data [11]. The storage cost is set to r_u = \$0.01133 per GB, which is equivalent to \$0.34 per GB per month [26] amortized on a daily basis assuming that data placement decisions are made at the beginning of each day. The computation cost is set to c_u = \$0.036 per GHz which is equivalent to \$0.00001 per GHz per second [27] assuming that each service lasts one hour spread over the day. Finally, the bandwidth cost is set to b_{uv} = \$0.009 per Mbps which is equivalent to \$0.02 per GB of transferred data [28]. We remark that our evaluation code is publicly available online in [29].

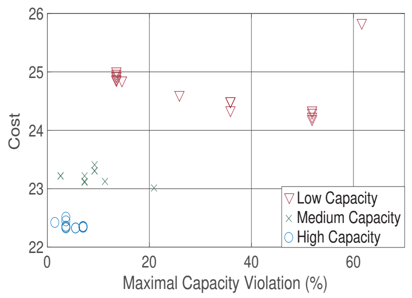
5.2 Evaluation Results

Throughout the evaluations, we consider three scenarios, each characterized by its storage, computation, and bandwidth capacities: (i) low capacity scenario with R_u = 100GBs, C_u = 20GHz, B_{uv} = 100Mbps; (ii) medium capacity scenario with R_u = 150GBs, C_u = 30GHz, B_{uv} = 150Mbps; and (iii) high capacity scenario with R_u = 200GBs, C_u = 40GHz, B_{uv} = 200Mbps. The capacity of the links inter-connecting the base stations with one another (X2 links in the figure) is always half of the capacity of the rest of the links.

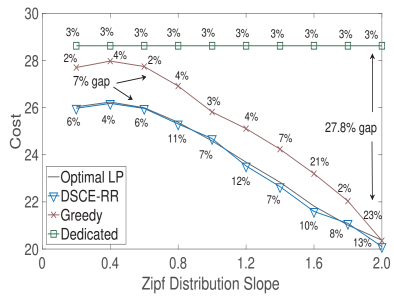
For each scenario, we compute the total resource cost (objective function value in (11)) and the maximal (worst) capacity violation across all nodes and links achieved by four algorithms: (i) the optimal solution to the linear relaxation of the DSCE problem (Optimal LP), (ii) the proposed approximation algorithm (DSCE-RR) that extracts a number of embeddings and randomly chooses one, and (iii)-(iv) two baseline schemes (Greedy and Dedicated) that represent the output of DSCE-RR algorithm when BSs are restricted to store the most popular data objects (intuitive naive solution) or when storage is regarded as a dedicated resource and thus sharing of data objects among services is not permitted (representative of state-of-the-art SCE methods).



(a) Cost & capacity violation in different scenarios.

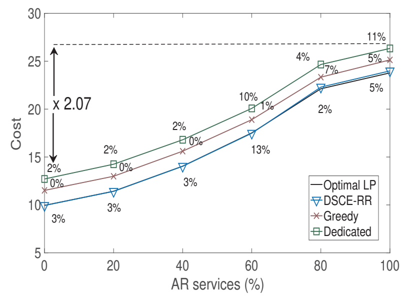


(b) Cost vs. capacity violation per extracted embedding.

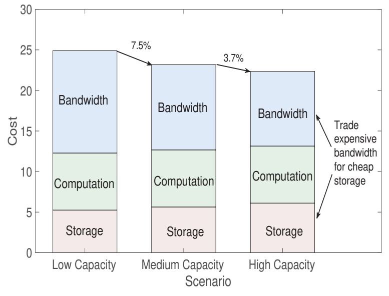


(c) Impact of demand distribution.

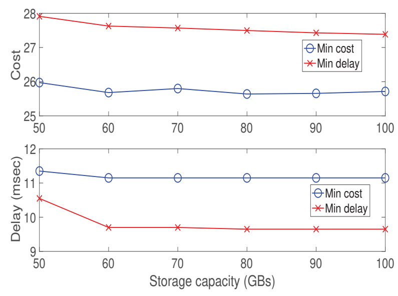
Figure 5: (a) Cost and maximal capacity violation achieved by Optimal LP, DSCE-RR, Greedy and Dedicated algorithms in low, medium and high capacity scenarios. (b) Scatter plot of cost versus maximal capacity violation across all the embeddings extracted by DSCE-RR algorithm. (c) Different slopes of the service demand distribution are examined in medium capacity scenario with labels representing the respective maximal capacity violation factors.



(a) Impact of different types of services.



(b) Cost breakdown.



(c) Cost vs. delay minimization.

Figure 6: (a) Cost for different portions of AR and VS services in medium capacity scenario. (b) Breakdown of cost into bandwidth, computation and storage components. (c) Impact of optimizing different objectives in low capacity scenario.

Figure 5a shows the results for each scenario and algorithm. Note that Optimal LP does not violate any of the resource capacities. However, the solution it returns may contain fractional values and therefore it only serves as a lower bound to the optimal integer solution in order to measure how far from optimal the other algorithms perform. While in all scenarios some capacities are violated, the violation factors are not significant. The maximal violation of DSCE-RR is 13.5% in the low capacity scenario. However, for the medium and high capacity scenarios DSCE-RR achieves the optimal cost and much smaller capacity violations (2.6% and 3.6%). We note that if we increase the capacities even more (e.g., by 25%), then all the violation factors become zero illustrating the fact that when capacities are high, there exists an Optimal LP solution that is, in fact, integer, and our proposed approximation algorithm is able to find it. Besides, *DSCE-RR performs better than the Greedy and Dedicated baseline schemes, with gains up to* 6.1% *and* 12%, *respectively.*

Figure 5b shows the scatter plot of cost and maximal capacity violation corresponding to all the embeddings extracted by DSCE-RR. A total of 17, 10 and 12 embeddings are extracted in the low, medium and high capacity scenarios, respectively. We notice that the values vary largely from one embedding to another, especially in the low capacity scenario where the maximal capacity violation can even exceed 60%. In most cases, however, *there is at least one embedding to choose that does not incur excessive capacity violation (less than* 3% *in two out of the three scenarios).*

Figure 5c examines different slopes of the Zipf probability distribution with which services are assigned to data objects. A higher slope represents a steeper distribution where service requests are concentrated on the same few data objects, and hence sharing of data among services is more likely to happen. On the other hand, a lower slope value represents a more shallow distribution of data objects. To eliminate the impact of random sizes of data objects, and focus instead on the demand distribution, we restrict all sizes to 10GBs in this experiment. We find that the cost achieved by all the algorithms but the Dedicated decreases with the slope value, illustrating the fact that *the effectiveness of the data placement decisions improves with the steepness of the demand distribution.* On the other hand, Dedicated is unable to exploit any of the data sharing opportunities and hence its performance does not improve with the slope. The gains of DSCE-RR are up to 7% and 27.8% over Greedy and Dedicated, respectively.

So far, we focused on a specific type of services, AR. We next examine a mixture of AR with video streaming (VS) services. The VS service graph differs from the AR one in that there is no real-time stream; the only stream comes from the access to the video stored in the storage function. Therefore, the bandwidth and accompanying computation requirements are lower than in the AR service. Figure 6a shows how the cost increases by more than a factor of 2 as we move from the pure VS (0% on the x axis) to the pure AR (100% on the x axis) operating point indicating the additional expenses required to support resource hungry AR services. We also note that DSCE-RR consistently outperforms the Greedy and Dedicated baseline schemes. The gains are higher at the pure VS point, up to 15.82% and 27.81%, respectively.

Next, we take a close look at the components that contribute to the overall cost. The cost breakdown in Figure 6b shows that storage, computation and bandwidth costs are comparable to each other where the largest contributor is the bandwidth cost. This was expected as the other resources (storage and computation) are usually available (or easy to deploy) to the network operator in larger quantities and at lower prices. As we increase the available capacities, the overall cost decreases, by 7.5% from the low to the medium capacity scenario, and by 3.7% more from the medium to the high capacity scenario. This is because for higher capacities, there exist more options for allocating resources to satisfy service requests at lower costs and the proposed DSCE-RR algorithm intelligently finds these options. Specifically, DSCE-RR places more data closer to the data demanding services, essentially *trading expensive bandwidth for cheaper storage resource units.*

Next, we explore the impact of our algorithm on other metrics such as delay. We can extend our algorithm to minimize delay by changing the values of the coefficients in the objective function (11). Specifically, we set c_u and r_u equal to zero for each node u and b_{uv} equal to the delay of each link (u, v) so that the new objective function represents the aggregate delay of the traffic streams in the service chains. Additional delay models may be used depending on the exact type of service, but these are out of scope of this paper. Figure 6c compares the cost and the delay of our algorithm when we apply it with cost (original function) or delay ($c_u = r_u = 0$ and $b_{uv} = 5$msec) as objective. As expected, when we minimize delay we can improve this metric more than when we minimize cost and vice versa. However, the two metrics are not conflicting; by optimizing one, benefits are also realized for the other metric.

We remark that the running time of DSCE-RR (implemented on a MacBook laptop with 2.3 GHz Core i5) is a few seconds in all the evaluations. Even if we increase the number of nodes or services by 10 times the running time remains of the same order indicating that DSCE-RR is not only near-optimal but also practical and scalable.

6 CONCLUSION

In this paper, we proposed and studied the SCE problem for services with intensive data requirements, where the same data can be shared by multiple service functions at possibly different locations. We provided a formulation that efficiently captures SCs' consumption of storage, computation, and bandwidth resources, and designed the first approximation algorithm for such problem class. The proposed algorithm is based on randomized rounding the linear relaxation of the original problem, where the service embeddings to be chosen with a given probability are extracted using a novel data-aware decomposition procedure, different from existing SCE methods. Evaluation results show that the proposed algorithm achieves near-optimal costs with up to 27.8% of the cost savings owed to the sharing of the data.

REFERENCES

[1] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, "Fog Computing and its Role in the Internet of Things", *MCC*, 2012.
[2] P. Mach, Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading'", *IEEE Communications Surveys & Tutorials., vol. 19, no. 3, pp. 1628-1656*, 2017.
[3] D. Bhamare, R. Jain, M. Samaka, A. Erbad, "A Survey on Service Function Chaining", *Journal of Network and Computer Applications, vol. 75, pp. 138-155*, 2016.
[4] M.S. Elbamby, C. Perfecto, M. Bennis, K. Doppler,"Towards Low-Latency and Ultra-Reliable Virtual Reality", *IEEE Network*, 2018.
[5] A. Hornes, S. Langer, "A Network Data Layer Concept for the Telco Industry", *White Paper, NGMN Alliance*, 2018.
[6] M. Rost, S. Schmid, "Virtual Network Embedding Approximations: Leveraging Randomized Rounding", *IFIP Networking*, 2018.
[7] M.A.T. Nejad, S. Parsaeefard, M.A. Maddah-Ali, T. Mahmoodi, B.H. Khalaj, "vS-PACE: VNF Simultaneous Placement, Admission Control and Embedding", *IEEE Journal on Selected Areas in Communications, vol. 36, no. 3, pp. 542-557*, 2018.
[8] J. Pei, P. Hong, K. Xue, D. Li, "Efficiently Embedding Service Function Chains with Dynamic Virtual Network Function Placement in Geo-distributed Cloud System", *IEEE Transactions on Parallel and Distributed Systems*, 2019.
[9] T. He, H. Khamfroush, S. Wang, T.L. Porta, S. Stein, "It's Hard to Share: Joint Service Placement and Request Scheduling in Edge Clouds with Sharable and Non-sharable Resources", *IEEE ICDCS*, 2018.
[10] J. Xu, L. Chen, P. Zhou, "Joint Service Caching and Task Offloading for Mobile Edge Computing in Dense Networks", *IEEE Infocom*, 2018.
[11] K. Poularakis, J. Llorca, A. Tulino, I. Taylor, L. Tassiulas, "Joint Service Placement and Request Routing in Multi-cell Mobile Edge Computing Networks", *IEEE Infocom*, 2019.
[12] A. Laghrissi and T. Taleb, "A Survey on the Placement of Virtual Resources and Virtual Network Functions", *IEEE Communications Surveys & Tutorials., vol. 21, no. 2, pp. 1409-1434*, 2019.
[13] Mengxue Liu, AndrÃa W. Richa, Matthias Rost, Stefan Schmid, "A Constant Approximation for Maximum Throughput Multicommodity Routing And Its Application to Delay-Tolerant Network Scheduling", *INFOCOM, pp. 46-54*, 2019.
[14] R. Cohen, L. Lewin-Eytan, J. S. Naor, D. Raz, "Near Optimal Placement of Virtual Network Functions", *IEEE Infocom*, 2015
[15] I. Benkacem, T. Taleb, M. Bagaa and H. Flinck, "Optimal VNFs Placement in CDN Slicing Over Multi-Cloud Environment", *IEEE Journal on Selected Areas in Communications, vol. 36, no. 3, pp. 616-627,* 2018.
[16] A. Laghrissi, T. Taleb, M. Bagaa, "Conformal Mapping for Optimal Network Slice Planning Based on Canonical Domains", *IEEE Journal on Selected Areas in Communications, vol. 36, no. 3, pp. 519-528*, 2018.
[17] B. Addis, D. Belabed, M. Bouet, S. Secci, "Virtual Network Functions Placement and Routing Optimization", *Cloudnet*, 2015.
[18] M. Barcelo, A. Correa, J. Llorca, A.M. Tulino, J.L. Vicario, A. Morell, "IoT-Cloud Service Optimization in Next Generation Smart Environments", *IEEE Journal on Selected Areas in Communications*, 2016.
[19] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On Dynamic Service Function Chain Deployment and Readjustment", *IEEE Transactions on Network and Service Management, vol. 14, no. 3, pp.543-553*, 2017.
[20] Pham, C., Tran, N.H., Ren, S., Saad, W., Hong, C.S., "Traffic-aware and Energy Efficient VNF Placement for Service Chaining: Joint Sampling and Matching Approach", *IEEE Trans. Services Computing*, 2017.
[21] S. Agarwal, F. Malandrino, C. F. Chiasserini, and S. De, "VNF Placement and Resource Allocation for the Support of Vertical Services in 5G networks", *IEEE/ACM Transactions on Networking*, 2019
[22] I. Baev, R. Rajaraman, C. Swamy, "Approximation Algorithms for Data Placement Problems", *SIAM Journal on Comp., vol. 38*, 2008.
[23] S. Borst, V. Gupta, A. Walid, "Distributed Caching Algorithms for Content Distribution Networks", *IEEE Infocom*, 2010.
[24] K. Shanmugam, N. Golrezaei, A. Dimakis, A. Molisch and G. Caire, "Femto-Caching: Wireless Content Delivery Through Distributed Caching Helpers", *IEEE Transactions on Information Theory, vol. 59, no. 12*, 2013.
[25] W. Hoeffding, "Probability Inequalities for Sums of Bounded Random Variables", *Journal of the American Statistical Association*, vol. 58, pp. 13-30, 1963.
[26] https://cloud.google.com/compute/disks-image-pricing
[27] https://rominirani.com/google-cloud-functions-tutorial-pricing-9cc6dc47f7c0
[28] https://cloud.google.com/compute/network-pricing
[29] https://www.dropbox.com/s/foya5jmchwxxdst/mobihoc20.m?dl=0