A Utility-based Optimization Framework for Edge Service Entity Caching

Yu Liang, Jidong Ge, *Member, IEEE,* Sheng Zhang, *Member, IEEE,* Jie Wu, *Fellow, IEEE,* Ze Tang, and Bin Luo, *Member, IEEE*

Abstract—Edge computing is one of the emerging technologies aiming to enable timely computation at the network edge. With virtualization technologies, the role of the traditional edge providers is separated into two: edge infrastructure providers (EIPs), who manage the physical edge infrastructure, and edge service providers (ESPs), who purchase slices of physical resources (e.g., CPU, bandwidth, memory space, disk storage) from EIPs and then cache service entities to offer their own value-added services to end users. When an ESP caches a service entity in an edge server, the ESP has to pay some fees (i.e, the cache cost) to the EIP that owns the edge server. One of the fundamental problems in edge virtualization is the so-called service entity caching problem, i.e., where to place service entities for an ESP to minimize the cache cost. In this paper, we study the service entity caching problem from the utility perspective. We use 'utility' to denote the positive impact on a client from caching a service entity in an edge server, and the exact meaning of utility can vary depending on specific scenarios. We formulate the Utility-based Service Entity Caching (UtilitySEC) problem, which can be generalized to many existing problems by modifying the 'utility'. We prove that the UtilitySEC problem is NP-complete and design an approximation algorithm for it. Extensive simulations are conducted to evaluate the performance of the proposed framework.

Index Terms—Edge computing, service entity caching, set cover, utility.

1 Introduction

The volume of the data generated at the edge of the Internet increases explosively in recent years. For example, Boeing 787 aircraft creates half a terabyte of data per flight [1]; a high-definition, traffic monitoring camera can generate several terabytes of data in a day [35].

Traditional centralized data processing technologies (e.g., cloud computing) are no longer able to handle such large amount of data efficiently, due to the following two reasons. On the one hand, linearly-increased computation capabilities in centralized compute clusters cannot match the data generated in the network edge with an exponential growth. On the other hand, transmitting such large amount of data to centralized compute clusters incurs a non-negligible latency.

Edge computing is one of the emerging technologies aiming to enable timely computation at the network edge [14, 20]. Many major cloud providers take edge computing as a promising paradigm to overcome the above issues. For example, over 500 edge servers are deployed in China by Alibaba [2]; Google has deployed more than 1,400 edge servers worldwide [4]. With geo-distributed edge servers, end users can have low-latency edge service anytime and anywhere, which greatly mitigates the effect of large amounts of data on backbone network and centralized data centers.

According to the Open Edge Computing initiative [3], edge cloud resources tend to be virtualized and can be allocated at a fine granularity by the aid of lightweight virtualization techniques [17]. This enables edge virtualization, a paradigm that decouples the functionalities in an edge environment by separating the role of the traditional edge providers into two: edge infrastructure providers (EIPs), who manage the physical edge infrastructure, and edge service providers (ESPs), who purchase slices of physical resources (e.g., CPU, bandwidth, memory space, disk storage) from EIPs and then *cache* service entities to offer their own value-added services to end users (EUs). Taking the Distributed Interactive Application (DIA) [36] atop edge computing for example. A DIA allows a group of distributed users to interact with each other synchronously via their mobile devices or computers. It usually consists of two components: service entity and client. A service entity maintains application metadata (including user state and application state), while a client (user) is only responsible for sending user-initiated operations to the service entities and receiving updates from the service entities.

When an ESP caches a service entity in an edge server, the ESP has to pay some fees (i.e, the cache cost) to the EIP that owns the edge server. One of the fundamental problems in edge computing is the so-called *service entity caching* problem, i.e., where to place service entities for an edge service provider to minimize the total cache cost.

- *Y. Liang, J.D. Ge, S. Zhang, Z. Tang, and B. Luo are with the State Key Laboratory for Novel Software Technology, Nanjing University, China. Y. Liang, J.D. Ge, Z. Tang, and B. Luo are also with the Software Institute, Nanjing University. S. Zhang is also with the Department of Compute Science and Technology, Nanjing University.*
  E-mail: {dg1832003, mf1832146}@smail.nju.edu.cn, {gjd, sheng, luobin}@nju.edu.cn.
- *J. Wu is with the Center for Networked Computing, Temple University, Philadelphia, PA 19122, USA.*
  E-mail: jiewu@temple.edu.

TABLE 1: Some typical problems in edge computing-related scenarios and the analogues of service entity, user/client, and utility.

Problems	Service Entity	User / Client	Utility$^\sharp$
IoT Application Provisioning [34]	an IoT application	an IoT data source	the bandwidth allocated to a data source along the path between an application and a data source
Server Placement in Probabilistic Networks [30]	a physical server	a client	the probability that a client successfully connects to a server
FemtoCaching [19]	a video file	a user terminal	the probability that a user terminal can access a video file

$^\sharp$the utility gained by a client from caching a single service entity in a server.

In this paper, we study the *service entity caching* problem from the utility perspective. We use 'utility' to denote the positive impact on a client from caching an entity in an edge server, and the exact meaning of utility can vary depending on specific scenarios. For example:

- IoT application provisioning [34]: an IoT (Internet-of-Things) application receives continuous data from one or more data sources and performs analysis on received data. In this case, an application can be seen as a service entity; an IoT data source can be seen as a client; and the utility gained by a data source from an IoT application is the bandwidth allocated to the source along the path between them.
- Server placement in probabilistic networks [30]: when the connections between mobile clients and edge servers are not reliable (i.e., wireless channels are lossy and unreliable, making the success of a transmission inherently probabilistic), the wireless connection service provider may want to ensure that the number of connected servers of a client is no less than a threshold. In this case, the probability that a client successfully connects to a server can be seen as the utility gained by a client in UtiltySEC from caching a single entity in a server in UtiltySEC .

We formulate the Utility-based Service Entity Caching (UtilitySEC) problem: given the locations of edge servers and the utility requirement of each user, an ESP must select some servers to place its service entities, so as to minimize the overall cache cost. We prove that the UtilitySEC problem is NP-complete and design an approximation algorithm for it. Extensive simulations are conducted to evaluate the performance of the proposed framework. Our main contributions are three-fold:

- We propose the UtilitySEC problem, which can be generalized to many existing problems by modifying the 'utility'.
- We design an efficient algorithm for UtilitySEC, and provide theoretical analysis on the approximation ratio.
- We evaluate the proposed algorithm using trace-driven simulations.

The rest of the paper is organized as follows. We motivate our study in Section 2. We introduce the UtilitySEC problem in Section 3. The NP-completeness result is presented in Section 4. We then present an approximation algorithm in Section 5. Evaluation is given in Section 6. We survey related work in Section 7 and conclude the paper in Section 8.

2 MOTIVATION

In this section, we motivate our study by showing many existing problems can be solved under the UtilitySEC optimization framework.

As we will shortly indicate in Section 3, the UtilitySEC optimization framework consists of three elements: *service entity*, *user/client*, and *utility*. Here, the 'utility' denotes the utility gained by a client from caching a single service entity in a server. In the following, we present several typical problems in edge computing-related scenarios in prior studies [19, 30, 34], and in each problem, we will explain what are the analogues of service entity, user/client, and utility. Main comparison results are summarized in Table 1.

IoT application provisioning [34]: an IoT application receives continuous data from one or more data sources and performs analysis on received data. In brief, the problem of IoT application provisioning is to decide where to place a set of IoT applications to satisfy the bandwidth requirements of each data source. In this problem, an IoT application can be seen as a service entity; an IoT data source can be seen as a client; and the bandwidth allocated to a data source along the path between an application and the data source can be seen as the utility gained by a client in UtiltySEC from caching a single service entity in a server in UtiltySEC .

Server placement in probabilistic networks [30]: when the connections between mobile clients and edge servers are not reliable, the service provider may want to ensure that the number of connected servers by a client is no less than a threshold. In this problem, a physical server can be seen as a service entity; and the probability that a client successfully connects to a server can be seen as the utility gained by a client in UtiltySEC from caching a single service entity in a server in UtiltySEC .

FemtoCaching [19]: in FemotoCaching, helpers (i.e., video content servers) store video files for user terminals, and the connectivity between users and helpers is a bipartite graph. Different user terminals may request different files with different probabilities. The problem of FemtoCaching is to decide which set of video files each helper should cache, so as to maximize the probability that each user terminal requests one of the video files that are accessible through its connected helpers. In this problem, a video file can be seen as a service entity; a user terminal can be seen as a client; and the probability that a user terminal can access a video file can be seen as the utility gained by a client from caching a single service entity in a server.

The comparison list is far from exhaustive. However, we believe these examples illustrate a few representative scenarios. Thus, solving the UtilitySEC problem in our study will help us develop techniques that hold across many similar scenarios.

3 The UtilitySEC Problem

Edge servers are usually deployed on a business premise such as in a doctor office or a coffee shop [18]. According to the Open Edge Computing initiative [3], edge server resources tend to be virtualized and can be allocated at a fine granularity by the aid of lightweight virtualization techniques. Therefore, in this paper, an edge service provider can rent edge resources from edge infrastructure providers to cache its own value-added entities.

We consider a metropolitan-area edge computing scenario which contains a set of N edge servers, denoted by $s_1, s_2, ...,$ and s_N. These servers are dispersed within a city, e.g., inside restaurants and in schools. Each edge server s_i is associated with a cache cost w_i, i.e., it costs w_i to cache a service entity in edge server s_i. There are M users/clients, $c_1, c_2, ..., c_M$, which are also dispersed in the same city.

Naturally, when we cache a service entity, it may have a nonnegative impact on each user/client. We use $u(s_i, c_j)$ to denote the utility gained by c_j if we cache a service entity in edge server s_i. Note that, $u(s_i, c_j)$ may not be a constant, and it can be explained in many different ways depending on specific scenarios. See Section 6 for more explanations.

The utilities gained by a client from multiple entities are assumed to be additive. For example, in the problem of IoT application provisioning [34], the bandwidths allocated to a data source along the path between the data source and multiple IoT applications are additive.

Each client has a utility requirement U. That is, the sum of the utilities gained by each client from the cached entities should be no less than U. It should be noted that the proposed algorithm can be easily extended to the case in which clients have heterogenous utility requirements by rewriting Eq. (1b) for each client.

We use x_i to indicate whether a service entity is placed/cached in edge server s_i, i.e., $x_i = 1$ if a service entity is placed in s_i, otherwise, it is 0. Main notations are summarized in Table 2.

The objective of the UtilitySEC problem is to minimize the total cache cost, such that the utility requirement of each client is satisfied. The UtilitySEC problem is formally defined as follows:

$$\min \quad \sum_{i=1}^{N} w_i x_i \qquad \text{[UtilitySEC]} \qquad (1a)$$

$$\text{s.t.} \quad \sum_{i=1}^{N} u(s_i, c_j) x_i \geq U, \qquad \forall j \in [1, M] \qquad (1b)$$

$$x_i \in \{0, 1\}, \qquad \forall i \in [1, N] \qquad (1c)$$

TABLE 2: Main notations for quick reference.

Symbol	Meaning
N	the number of edge servers
s_i	the i-th edge server
w_i	the cache cost in server s_i
M	the number of users/clients
c_j	the j-th user/client
$u(s_i, c_j)$	the utility gained by c_j if we cache a service entity in s_i
U	the utility requirement
x_i	cache a service entity in s_i or not
g_{ij}	the start index of the utilities gained by c_j from caching a service entity in s_i
d_{ij}	the actual utility obtained by c_j from caching a service entity in s_i

Eq. (1b) ensures that the total utilities obtained by each client is no less than the requirement; Eq. (1c) is the integral constraint. In the next two sections, we will show the UtilitySEC problem is NP-complete and propose an approximation algorithm for it, respectively.

4 NP-completeness Result

By reducing the NP-complete Set Cover (SC) problem [24] to UtilitySEC, we can prove that the decision version of UtilitySEC is NP-complete.

***Theorem** 1:* The decision version of UtilitySEC is NP-complete.

Proof: We provide the descriptions of the decision version of SC and UtilitySEC as follows.

- *Decision version of SC*: Given a universe $\mathcal{H} = \{e_1, e_2, ..., e_m\}$ of m elements and an integer q, a collection of subsets of $\mathcal{H}$, $\mathcal{R}_1$, $\mathcal{R}_2$, ..., and $\mathcal{R}_n$, does there exist a sub-collection of these subsets with size no more than q that covers all elements of $\mathcal{H}$?
- *Decision version of UtilitySEC*: Given a set of users $c_1, c_2, ..., c_M$, a set of edge servers $s_1, s_2, ..., s_N$ with cache costs $w_1, w_2, ..., w_N$, respectively, and the utility requirement U. The utility obtained by c_j from caching a service entity in s_i is $u(s_i, c_j)$. Does there exist a service entity caching solution that incurs a cache cost no more than a threshold W?

Without loss of generality, let

$$< m, n, q, \mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_N >$$

denote an instance of SC; let

$$< M, N, w_1, ..., w_N, U, u(s_1, c_1), ..., u(s_N, c_M), W >$$

denote an instance of UtilitySEC.

In the following, we show that, any instance of SC can be polynomially reduced to an instance of UtilitySEC. The reduction maps an instance of SC into an instance of UtilitySEC using the following *rules*:

[1], $M \leftarrow m$; [2], $N \leftarrow n$; [3], for each cache cost w_i, $w_i \leftarrow 1$; [4], $W \leftarrow q$; [5], $U \leftarrow 1$; [6], for each utility $u(s_i, c_j)$, $u(s_i, c_j) \leftarrow 1$ *if and only if* $e_j \in R_i$ in the instance of SC.

Rules [1] and [2] specify the number of clients and edge servers, respectively. Rule [3] sets each cache cost to 1. Rule [4] means the number of cached service entities should not exceed q. Rule [5] sets the utility requirement to 1. The last rule sets each $u(s_i, c_j)$ to either 1 or 0, depending on whether subset R_i contains element e_j. It is easy to see that, the reduction from SC to UtilitySEC can be finished in polynomial time.

To confirm the validity of this reduction, we have to show that it works in the case of either outcome depicted.

On the one hand, if the instance of SC has a yes solution, i.e., there exist q subsets among $\mathcal{R}_1$, $\mathcal{R}_2$, ..., and $\mathcal{R}_N$ that can cover all m elements. According to the reduction rules, in the corresponding UtiltiySEC instance, we can cache W service entities in W servers that make each client obtain no less than $U = 1$ utility.

On the other hand, if the instance of SC does not have a yes solution, i.e., it needs at least $(q+1)$ subsets to cover all m elements. According to the reduction rules, in the corresponding UtiltiySEC instance, we also need to cache at least $(W+1)$ service entities to satisfy the utility requirement of each client.

Finally, it is easy to verify the decision version of UtilitySEC is in NP, hence, the theorem is proven. □

Simplifying UtilitySEC. Before presenting our algorithm, we show that we can safely assume that all $u(s_i, c_j)$'s and U are integral.

Without loss of generality, all $u(s_i, c_j)$'s and U are rational numbers. Since they are rational numbers, we can rewrite each of them as a fraction, i.e., $u(s_i, c_j)$ can be rewritten as $\frac{a_{ij}}{b_{ij}}$ and U can be rewritten as $\frac{a}{b}$, where a_{ij}, b_{ij}, a, and b are integrals. Then, the constraint (1b) in the UtilitySEC problem becomes

$$\sum_{i=1}^{N} \frac{a_{ij}}{b_{ij}} x_i \geq \frac{a}{b}, \tag{2}$$

which is equivalent to

$$\sum_{i=1}^{N} (a_{ij} b \prod_{k \neq i} b_{kj}) x_i \geq a \prod_{k=1}^{N} b_{kj}, \tag{3}$$

where both $(a_{ij} b \prod_{k \neq i} b_{kj})$ and $(a \prod_{k=1}^{N} b_{kj})$ are integrals. Therefore, we can safely assume that all $u(s_i, c_j)$'s and U are integral.

5 The Proposed Algorithm and Approximation Ratio Analysis

We present a greedy algorithm for UtilitySEC and its performance analysis in this section.

5.1 The Greedy Algorithm (GA) for UtilitySEC

The main idea of GA is as follows. GA consists of multiple iterations. In each iteration, we compute the cost-effectiveness of each edge server in which no service entity is placed yet; we then select the edge server that has the smallest cost-effectiveness to cache a new entity. Here, the 'cost-effectiveness' of an edge server is defined as the ratio of its cache cost to the marginal utilities it brings to all clients. The details are shown in Alg. 1.

Algorithm 1: Greedy Algorithm (GA) for UtilitySEC

```
Input: $w_i$ for every $i \in [1, N]$, $u(s_i, c_j)$ for every pair of $i \in [1, N]$ and $j \in [1, M]$, $U$
Output: $x_i$ for every $i \in [1, N]$
1  $\forall i \in [1, N]$, $x_i \leftarrow 0$;
2  $C \leftarrow \{c_1, c_2, ..., c_M\}$;
3  while $C \neq \emptyset$ do
4      $ce\_m \leftarrow \sum_{k=1}^{N} w_k$;
5      $ce\_index \leftarrow -1$;
6      for $i = 1; i \leq N; i++$ do
7          if $x_i = 0$ then
8              $ce \leftarrow \frac{w_i}{\sum_{c_j \in C} \min\{u(s_i, c_j), U - \sum_{k=1}^{N} u(s_k, c_j) x_k\}}$;
9              if $ce < ce\_m$ then
10                 $ce\_m \leftarrow ce$;
11                 $ce\_index \leftarrow i$;
12     $x_{ce\_index} \leftarrow 1$;
13     for each $c_j \in C$ do
14         if $\sum_{i=1}^{N} u(s_i, c_j) x_i \geq U$ then
15             remove $c_j$ from $C$, i.e., $C \leftarrow C \setminus \{c_j\}$;
16 return $x_1, x_2, ..., x_N$
```

Initially, no service entity is placed (line 1); we use C to denote the set of clients that have not obtained enough utilities (line 2). Remember that the utility requirement of each client is U; when the sum of utilities obtained by some client c_j from cached service entities, which is $\sum_{k=1}^{N} u(s_k, c_j) x_k$, exceeds the requirement, the additional utilities (i.e., $\sum_{k=1}^{N} u(s_k, c_j) x_k - U$) are useless.

In each iteration, for each edge server s_i, if we have not cached a service entity in edge server s_i, we compute its cost-effectiveness (line 8) as follows:

$$ce \leftarrow \frac{w_i}{\sum\limits_{c_j \in C} \min\{u(s_i, c_j), U - \sum\limits_{k=1}^{N} u(s_k, c_j) x_k\}}. \tag{4}$$

In Eq. (4), $\min\{u(s_i, c_j), U - \sum_{k=1}^{N} u(s_k, c_j) x_k\}$ denotes the utilities obtained by c_j if we cache a service entity in edge server s_i. The readers may wonder why the second term is $(U - \sum_{k=1}^{N} u(s_k, c_j) x_k)$. The reason is simple; the set C maintains the clients that have not received enough utilities and c_j belongs to C, thus, $\sum_{k=1}^{N} u(s_k, c_j) x_k$ is less than U but $\sum_{k=1}^{N} u(s_k, c_j) x_k + u(s_i, c_j)$ may be larger than U. Before proceeding to the next iteration, we cache a service entity in the server that has the smallest cost-effectiveness (line 12) and update the set C (lines 13-15).

Complexity. The time complexity of GA is dominated by the while loop. Since there are N edge servers, the while loop contains at most N iterations. In each iteration, GA has to compute the cost-effectiveness of each edge server that has not cached any service entity yet, which requires $O(MN)$ time; updating the set C also requires $O(MN)$ time. Combining them together, the time complexity of GA is $O(N^2M)$.

Example. We provide an example to better illustrate the GA algorithm. There are 4 edge servers that have heterogeneous cache costs and 5 clients. The utility requirement U of each client is 3. Tab. 3 shows the details of the utilities. We now check how GA works.

Initially, no service entity is placed. In the first iteration of the while-loop (lines 3-15), GA computes the cost-effectiveness of each server according to Eq. (4): $ce_1 = \frac{8}{5}$, $ce_2 = \frac{6}{6} = 1$, $ce_3 = \frac{9}{6} = \frac{3}{2}$, and $ce_4 = \frac{14}{7} = 2$. GA chooses to place a service entity in server s_2 since ce_2 is the smallest. Since not all clients meet their utility requirements, GA enters into the second iteration of the while-loop, GA computes the cost-effectiveness of s_1, s_3, and s_4 as follows: $ce_1 = \frac{8}{5}$; $ce_3 = \frac{3}{2}$; for s_4, although it can bring 2 units of utilities to c_1, the actual utility it brings to c_1 is $\min\{2, U-2\} = 1$, therefore, $ce_4 = \frac{14}{6} = \frac{7}{3}$. GA chooses s_3 to cache a service entity. Again, since c_3 and c_4 have not met their utility requirements, GA enters into the third iteration. Due to the same reason, we have $ce_1 = \frac{8}{3}$ and $ce_4 = \frac{14}{2} = 7$. GA chooses s_1 to place an entity and all clients satisfy their utility requirements.

5.2 Analysis

In this subsection, we analyze the approximation ratio of the proposed algorithm using the dual fitting technique. We first provide the analysis overview, then we present a few preliminaries that are key to analysis, and finally we present the details of the analysis.

5.2.1 Overview of Analysis

Denote the optimum value of UtilitySEC by OPT. In order to find the approximation ratio of GA, we have to find the lowerbound of OPT.

The original UtilitySEC problem is a mixed integer linear program (MILP). We relax the integral constraint (i.e., Eq. (1c)) and have the following problem:

$$\begin{aligned}
\min \quad & \sum_{i=1}^{N} w_i x_i && \text{[pfUtilitySEC]} && (5a)\\
\text{s.t.} \quad & \sum_{i=1}^{N} u(s_i, c_j) x_i \geq U, && \forall j \in [1, M] && (5b)\\
& x_i \geq 0, && \forall i \in [1, N] && (5c)\\
& -x_i \geq -1, && \forall i \in [1, N] && (5d)
\end{aligned}$$

Under the dual fitting technique, the above problem is called the primal fractional problem, thus, we denote it by pfUtilitySEC. Denote the optimum value of pfUtilitySEC by OPT_f. Due to the relaxation, we have

$$OPT_f \leq OPT. \tag{6}$$

TABLE 3: An example that contains 4 edge servers and 5 clients. The utility requirement U is 3.

Cache Cost	Client / Server	c_1	c_2	c_3	c_4	c_5
8	s_1	1	1	2	1	0
6	s_2	2	2	0	1	1
9	s_3	1	1	1	1	2
14	s_4	2	1	1	1	2

Using the purely mechanical procedure for obtaining the dual of a linear problem, we have the dual problem of pfUtilitySEC:

$$\begin{aligned}
\max \quad & \sum_{j=1}^{M} U y_j - \sum_{i=1}^{N} z_i && \text{[dfUtilitySEC]} && (7a)\\
\text{s.t.} \quad & \sum_{j=1}^{M} u(s_i, c_j) y_j - z_i \leq w_i, && \forall i \in [1, N] && (7b)\\
& y_j \geq 0, && \forall j \in [1, M] && (7c)\\
& z_i \geq 0, && \forall i \in [1, N] && (7d)
\end{aligned}$$

This problem is denoted as dfUtilitySEC. According to the weak duality theorem [24], any feasible solution to dfUtilitySEC is no larger than any feasible solution to pfUtilitySEC, that is, given any feasible solution $(\mathbf{y}, \mathbf{z})$ to dfUtilitySEC and any feasible solution $\mathbf{x}$ to pfUtilitySEC, we have

$$\sum_{j=1}^{M} U y_j - \sum_{i=1}^{N} z_i \leq \sum_{i=1}^{N} w_i x_i. \tag{8}$$

Since the optimum solution to pfUtilitSEC is also a feasible solution to pfUtilitySEC, we have

$$\sum_{j=1}^{M} U y_j - \sum_{i=1}^{N} z_i \leq OPT_f. \tag{9}$$

Combining Eqs. (6) and (9) together, we know any feasible solution to dfUtilitySEC is a lowerbound of OPT.

Our analysis can be briefly summarized in the following three steps.

Step 1, find a dual solution $(\mathbf{y}', \mathbf{z}')$ that fully pays the primal solution $\mathbf{x}$ generated by GA, that is,

$$\sum_{j=1}^{M} U y'_j - \sum_{i=1}^{N} z'_i \geq \sum_{i=1}^{N} w_i x_i. \tag{10}$$

It should be noted that $(\mathbf{y}', \mathbf{z}')$ may not be dual feasible.

Step 2, scale down $(\mathbf{y}', \mathbf{z}')$ by a factor of λ and get a dual feasible solution $(\mathbf{y}'', \mathbf{z}'')$, that is,

$$\sum_{j=1}^{M} u(s_i, c_j) y''_j - z''_i \leq w_i, \forall i \in [1, N]. \tag{11}$$

Step 3, prove that λ is the approximation ratio.

5.2.2 Preliminaries

Before analyzing GA, a few necessary concepts and lemmas are explained below.

TABLE 4: The $cost(c_j, k)$'s in the solution produced by running GA on the example shown in Tab. 3.

	c_1	c_2	c_3	c_4	c_5
$cost(c_j, 1)$	1	1	$\frac{3}{2}$	1	1
$cost(c_j, 2)$	1	1	$\frac{8}{3}$	$\frac{3}{2}$	$\frac{3}{2}$
$cost(c_j, 3)$	$\frac{3}{2}$	$\frac{3}{2}$	$\frac{8}{3}$	$\frac{8}{3}$	$\frac{3}{2}$

A client is said to be "alive" during the execution of GA, if its utility requirement has not been satisfied yet.

When a server s_i is going to be selected by the GA algorithm to cache a service entity, the cache cost w_i is distributed among the alive clients as follows. For each alive client c_j, the sum of utilities it obtains from cached service entities (except s_i) is $\sum_{k=1,k\neq i}^{N} u(s_k, c_j)x_k$. We denote this value by g_{ij}. Since c_j is alive before s_i caches a service entity, we have $g_{ij} < U$. Hence, the utility gained by c_j from caching a service entity in s_i is $\min\{u(s_i, c_j), U - g_{ij}\}$. For simplicity of presentation, let

$$d_{ij} = \min\{u(s_i, c_j), U - g_{ij}\}, \tag{12}$$

and d_{ij} is the actual utility obtained by c_j from caching a service entity in s_i. Taking Tab. 3 for example, in the second iteration of the while-loop, when computing the cost-effectiveness of caching an entity in s_4, the sum of utilities c_1 obtains from the cached service entities is 2, i.e., $g_{41} = 2$; the actual utility obtained by c_1 from caching an entity in s_4 is $d_{41} = \min\{u(s_4, c_1), U - g_{41}\} = \min\{2, 3 - 2\} = 1$.

The cost-effectiveness defined in Eq. (4) can be seen as the cost we need to pay for getting one unit of utility. Define $cost(c_j, k)$ as the cost of c_j getting its k-th unit of utility, where $k \in [1, U]$. Since client c_j obtains its $(g_{ij}+1)$-th, $(g_{ij} + 2)$-th, ..., and $(g_{ij} + d_{ij})$-th utility from caching a service entity in edge server s_i, we have

$$cost(c_j, g_{ij} + 1) = \text{cost-effectiveness of } s_i, \tag{13}$$

$$cost(c_j, g_{ij} + 2) = \text{cost-effectiveness of } s_i, \tag{14}$$

$$\ldots\ldots$$

$$cost(c_j, g_{ij} + d_{ij}) = \text{cost-effectiveness of } s_i. \tag{15}$$

Taking Tab. 3 for example, client c_4 obtained its 1st, 2nd, and 3rd utilities from caching entities in s_2, s_3, and s_1, respectively: $cost(c_4, 1) = 1$, $cost(c_4, 2) = \frac{3}{2}$, and $cost(c_4, 3) = \frac{8}{3}$. The $cost(c_j, k)$'s in the solution produced by running GA on the example shown in Tab. 3 are provided in Tab. 4.

Based on the definition of $cost(c_j, k)$ and the greedy heuristic of GA, we have the following lemmas.

***Lemma** 1:* Assuming x_1, x_2, ..., and x_N is a feasible solution to the UtilitySEC problem, then

$$\sum_{j=1}^{M}\sum_{k=1}^{U} cost(c_j, k) = \sum_{i=1}^{N} w_i x_i.$$

Proof: When a server s_i is selected to cache an entity, it brings d_{ij} utilities to client c_j if c_j has not received enough utilities at that time point, and the cache cost w_i is equally distributed among these utilities it brings. □

***Lemma** 2:* For each client c_j, we have

$$cost(c_j, 1) \le cost(c_j, 2) \le \ldots\ldots \le cost(c_j, U).$$

Proof: Remember that GA is a greedy algorithm; in each iteration, it selects the edge server that has the smallest cost-effectiveness to cache a service entity. Because of this greedy nature, for any client, the cost of obtaining one unit of utility cannot decrease. □

5.2.3 Details of Analysis

Step 1, finding a dual solution $(\mathbf{y}', \mathbf{z}')$ that fully pays the primal solution $\mathbf{x}$ generated by GA. We now construct $(\mathbf{y}', \mathbf{z}')$ as follows.

For each $j \in [1, M]$, let

$$y'_j = cost(c_j, U). \tag{16}$$

For each $i \in [1, N]$, let

$$z'_i = \begin{cases} \sum_{j=1}^{M} (d_{ij} cost(c_j, U) - \sum_{k=g_{ij}+1}^{g_{ij}+d_{ij}} cost(c_j, k)) \\ \qquad\qquad\qquad \text{if } s_i \text{ caches a service entity,} \\ 0 \qquad\qquad\quad \text{otherwise.} \end{cases} \tag{17}$$

Remember that g_{ij} is the start index of the utilities gained by c_j from caching a service entity in s_i.

We have the following theorem.

***Theorem** 2:* The solution $\mathbf{x}$ generated by GA is fully paid by the dual solution $(\mathbf{y}', \mathbf{z}')$ in Eqs. (16) and (17).

Proof: According to Lemma 1, the solution $\mathbf{x}$ generated by GA is

$$\sum_{i=1}^{N} w_i x_i = \sum_{j=1}^{M}\sum_{k=1}^{U} cost(c_j, k).$$

The dual solution $(\mathbf{y}', \mathbf{z}')$ in Eqs. (16) and (17) is

$$\begin{aligned}
&\sum_{j=1}^{M} U y'_j - \sum_{i=1}^{N} z'_i \\
=&\sum_{j=1}^{M} U cost(c_j, U) - \sum_{i=1}^{N}\sum_{j=1}^{M}(d_{ij} cost(c_j, U) - \sum_{k=g_{ij}+1}^{g_{ij}+d_{ij}} cost(c_j, k)) \\
=&(\sum_{j=1}^{M} U cost(c_j, U) - \sum_{i=1}^{N}\sum_{j=1}^{M} d_{ij} cost(c_j, U)) \\
&+ \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=g_{ij}+1}^{g_{ij}+d_{ij}} cost(c_j, k)) \\
=&(\sum_{j=1}^{M} U cost(c_j, U) - \sum_{j=1}^{M}(\sum_{i=1}^{N} d_{ij}) cost(c_j, U)) \\
&+ \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=g_{ij}+1}^{g_{ij}+d_{ij}} cost(c_j, k)) \\
=&\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=g_{ij}+1}^{g_{ij}+d_{ij}} cost(c_j, k) = \sum_{j=1}^{M}(\sum_{i=1}^{N}\sum_{k=g_{ij}+1}^{g_{ij}+d_{ij}}) cost(c_j, k) \\
=&\sum_{j=1}^{M}\sum_{k=1}^{U} cost(c_j, k) \ge \sum_{i=1}^{N} w_i x_i.
\end{aligned}$$

The theorem follows immediately. □

Step 2, finding a dual feasible solution $(\mathbf{y}'', \mathbf{z}'')$ by scaling down $(\mathbf{y}', \mathbf{z}')$. Let u_{max} be $\max_{i,j} u(s_i, c_j)$ and H_M be $(1+\frac{1}{2}+\cdots+\frac{1}{M})$. We now construct $(\mathbf{y}'', \mathbf{z}'')$ by scaling down $(\mathbf{y}', \mathbf{z}')$ by a factor of $u_{max}H_M$.

For each $j \in [1, M]$, let

$$y''_j = \frac{y'_j}{u_{max}H_M}. \tag{18}$$

For each $i \in [1, N]$, let

$$z''_i = \frac{z'_i}{u_{max}H_M}. \tag{19}$$

***Theorem** 3:* $(\mathbf{y}'', \mathbf{z}'')$ defined Eqs. (18) and (19) is a feasible solution to dfUtilitySEC.

Proof: It is sufficient to show that $(\mathbf{y}'', \mathbf{z}'')$ satisfy the three constraints in Eq. (7). Obviously, they are nonnegative. Thus, in the following, we show they satisfy the first constraint, i.e., constraint (7b).

Consider any edge server s_i, without loss of generality, we assume caching a service entity in s_i brings a positive utility to c_1, c_2, ..., and c_{M_i}, where M_i is no larger than M. That is, for each $j \in [1, M_i]$, $u(s_i, c_j) \geq 1$; for each $j \in [M_i + 1, M]$, $u(s_i, c_j) = 0$.

Again, without loss of generality, we assume the GA algorithm makes c_1, c_2, ..., c_{M_i} stop being alive in the order, breaking the ties arbitrarily.

Case 1: s_i caches no service entity at the end of the execution of the GA algorithm, i.e., $x_i = 0$.

When the GA algorithm is about to enable a client c_j ($j \in [1, M_i]$) to gain its last unit of utility, i.e., the utility requirement of c_j would be satisfied if it gains one more unit of utility, s_i can make at least $(M_i - j + 1)$ clients obtain positive utilities if caching a service entity in s_i. This is because the GA algorithm makes c_1, c_2, ..., c_{M_i} stop being alive in the order. Since the GA algorithm is greedy, we have

$$cost(c_i, U) \leq \frac{w_i}{\sum_{k=j}^{M_i} d_{ik}} \leq \frac{w_i}{M_i - j + 1}. \tag{20}$$

The second inequality holds because $d_{ik} \geq 1$.

According to Eq. (17), $z''_i = 0$. We have

$$\begin{aligned}
&\sum_{j=1}^{M} u(s_i, c_j) y''_j - z''_i \\
=&\sum_{j=1}^{M_i} u(s_i, c_j) \frac{cost(c_j, U)}{u_{max}H_M} + \sum_{j=M_i+1}^{M} 0 \frac{cost(c_j, U)}{u_{max}H_M} - 0 \\
\leq&\frac{1}{H_M} \sum_{j=1}^{M_i} \frac{u(s_i, c_j)}{u_{max}} \frac{w_i}{M_i - j + 1} \\
=&\frac{w_i}{H_M} \sum_{j=1}^{M_i} \frac{1}{M_i - j + 1} = \frac{w_i}{H_M} \sum_{j=1}^{M_i} \frac{1}{j} = w_i.
\end{aligned}$$

Case 2: s_i caches a service entity at the end of the algorithm, i.e., $x_i = 1$.

Before the GA algorithm decides to cache a service entity in s_i, suppose that M' clients have already gained U units of utilities ($0 \leq M' \leq M_i - 1$). Then, we have

$$\begin{aligned}
&\sum_{j=1}^{M} u(s_i, c_j) y''_j - z''_i \\
=&\frac{1}{u_{max}H_M} (\sum_{j=1}^{M_i} u(s_i, c_j) cost(c_j, U) \\
&- \sum_{j=M'+1}^{M_i} (d_{ij} cost(c_j, U) - \sum_{k=g_{ij}+1}^{g_{ij}+d_{ij}} cost(c_j, k))) \\
=&\frac{1}{u_{max}H_M} (\sum_{j=1}^{M'} u(s_i, c_j) cost(c_j, U) \\
&+ (\sum_{j=M'+1}^{M_i} u(s_i, c_j) cost(c_j, U) - \sum_{j=M'+1}^{M_i} d_{ij} cost(c_j, U)) \\
&+ \sum_{j=M'+1}^{M_i} \sum_{k=g_{ij}+1}^{g_{ij}+d_{ij}} cost(c_j, k)) \\
\leq&\frac{1}{u_{max}H_M} (\sum_{j=1}^{M'} u(s_i, c_j) cost(c_j, U) \\
&+ \sum_{j=M'+1}^{M_i} \sum_{k=g_{ij}+1}^{g_{ij}+d_{ij}} cost(c_j, k)) \\
=&\frac{1}{H_M} (\sum_{j=1}^{M'} \frac{u(s_i, c_j)}{u_{max}} \frac{w_i}{M_i - j + 1} + \frac{1}{u_{max}} w_i) \\
\leq&\frac{w_i}{H_M} (\frac{1}{M_i} + \frac{1}{M_i - 1} + \cdots + \frac{1}{M_i - M' + 1} + \frac{1}{u_{max}}) \\
\leq&\frac{w_i}{H_M} (\frac{1}{M_i} + \frac{1}{M_i - 1} + \cdots + \frac{1}{M_i - M' + 1} + 1) \leq w_i.
\end{aligned}$$

Hence, $(\mathbf{y}'', \mathbf{z}'')$ is a feasible solution to dfUtilitySEC. □

Step 3, finding the approximation ratio. We now show the scaling factor $(H_M \cdot u_{max})$ in Step 2 is exactly the approximation ratio of the proposed algorithm.

***Theorem** 4:* GA is an $(H_M \cdot u_{max})$ factor approximation algorithm for UtilitySEC.

Proof:

$$\begin{aligned}
&\sum_{i=1}^{N} x_i w_i = \sum_{j=1}^{M} \sum_{k=1}^{U} cost(c_j, k) &&\text{/*due to Lemma 1*/} \\
\leq&\sum_{j=1}^{M} U y'_j - \sum_{i=1}^{N} z'_i &&\text{/*due to Theorem 2*/} \\
=&H_M u_{max} \sum_{j=1}^{M} U y''_j - \sum_{i=1}^{N} z''_i &&\text{/*due to scaling*/} \\
\leq&H_M u_{max} \cdot OPT_f &&\text{/*due to Theorem 3*/} \\
\leq&H_M u_{max} \cdot OPT. &&\text{/*due to relaxation*/}
\end{aligned}$$

□

6 Performance Evaluation

In this section, we evaluate the proposed algorithm using trace-driven simulations.

6.1 Dataset and Setup

We consider a metropolitan area that contains edge servers and users. Similar to a previous study [26], we used Starbucks' locations as the locations of edge servers, because the distribution of them in a city usually achieves a decent coverage of users, making them very suitable for placing edge servers. Therefore, we collected the locations of Starbucks within the 4th ring road of Beijing, China, and the Manhattan island of New York, US, as shown in Fig. 1. For each city, we calculated the minimum bounding rectangle of the Starbucks with two sides parallel to a meridian. Then, we extended this rectangle by adding 5km to each side, so as to form the area of interest. For locations of clients, we randomly generate them inside the area.

Since the number of edge servers was fixed, i.e., 92 in the Beijing trace and 201 in the New York trace, we cannot evaluate the proposed algorithm with more edge servers. We also conducted another set of simulations where the locations of both edge servers and users were randomly generated within an area.

Similar to [26], the delay between an edge server and a client is proportional to the Euclidean distance between them in all settings (Beijing, New York, or Synthetical). In our simulations, we assume 1km incurs 1ms. Therefore, when the locations of clients are known, we can calculate the delay between every pair of servers and clients. The average of these delays is denoted by $AvgDelay$. Similar to previous works [26, 30], the utility requirement of each client is generated randomly following a uniform distribution with the mean of 8; the average cost of caching a service entity in an edge server is also generated randomly following a uniform distribution with the mean of 3.

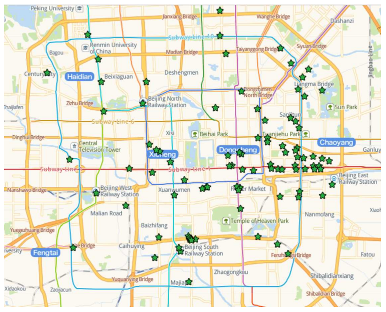
We consider two settings of utilities in our simulations.

Bandwdith-as-utility. In this (simple) setting, the utility $u(s_i, c_j)$ obtained by a client c_j from caching a service entity in edge server s_i is the bandwidth allocated along the connection path between s_i and c_j.

Delay-function-as-utility. In this (complex) setting, the utility $u(s_i, c_j)$ obtained by a client c_j from caching a service entity in edge server s_i is a non-negative, non-increasing function of the delay between s_i and c_j. In our simulations, we take $u(s_i, c_j) = \frac{AvgDelay}{(\text{delay between } s_i \text{ and } c_j)^{skew}}$, where $skew$ reflects the sensitiveness to delay. The impact of $skew$ is also evaluated in our simulations.

We compare GA with the following three algorithms: Incrementally Cache Algorithm (ICA), Randomly Cache Algorithm (RCA), and OPT.

ICA considers the utility requirement of each client one by one. ICA tries to satisfy the utility requirement of the first client by caching some service entities with minimum cache cost; then, ICA tries to satisfy the utility requirement of the second client by incrementally caching more service entities, and so on. RCA randomly chooses an edge server to cache a service entity, until the utility requirement of each client is satisfied. OPT is a brute-force algorithm that searches the best from a total of 2^N different solutions. The time complexity of OPT is extremely large, making it possible to run it on small settings where both of the number of edge servers and the number of clients are not large.


(a) Beijing, China

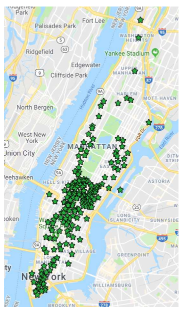

(b) New York, US

Fig. 1: Locations of 92 and 201 Starbucks in Beijing and New York, respectively.

6.2 Results

In this subsection, we show the simulation results on the Beijing, New York, and Synthetical traces with two different utility settings. All the results are obtained by averaging 10 independent runs.

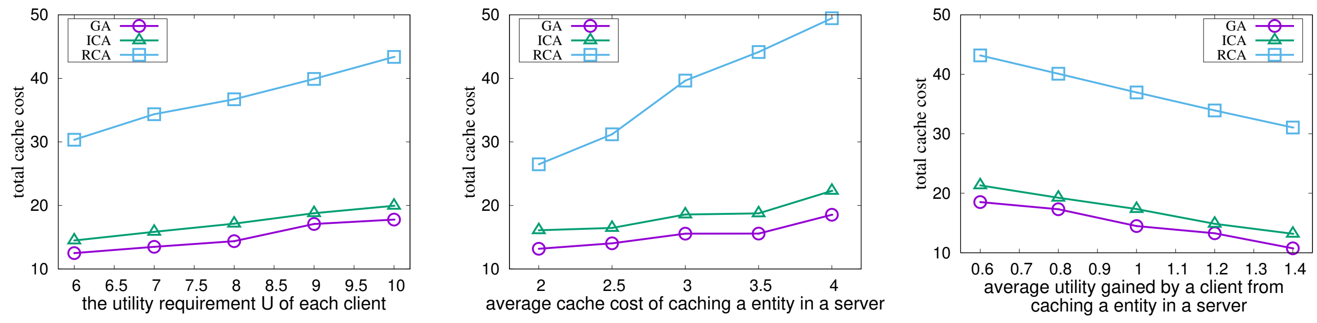
6.2.1 *Impact of Utility Requirement, Average Cache Cost, and Utility Settings*

Figs. 2 and 3 show the simulation results on the Beijing trace with the bandwidth-as-utility and delay-function-as-utility settings, respectively. Figs. 4 and 5 show the simulation results on the New York trace with the bandwidth-as-utility and delay-function-as-utility settings, respectively.

In general, GA achieved the smallest cache cost among GA, ICA, and RCA in these figures. GA caches service entities one by one, and whenever GA selects an edge server to place a service entity, GA takes all the clients into consideration. In contrast, ICA considers service entity caching from the perspective of the clients; ICA satisfies the utility requirement of each client one by one. When ICA selects an edge server to place a service entity, it takes only a part of the clients into account, which makes it perform worse than GA.

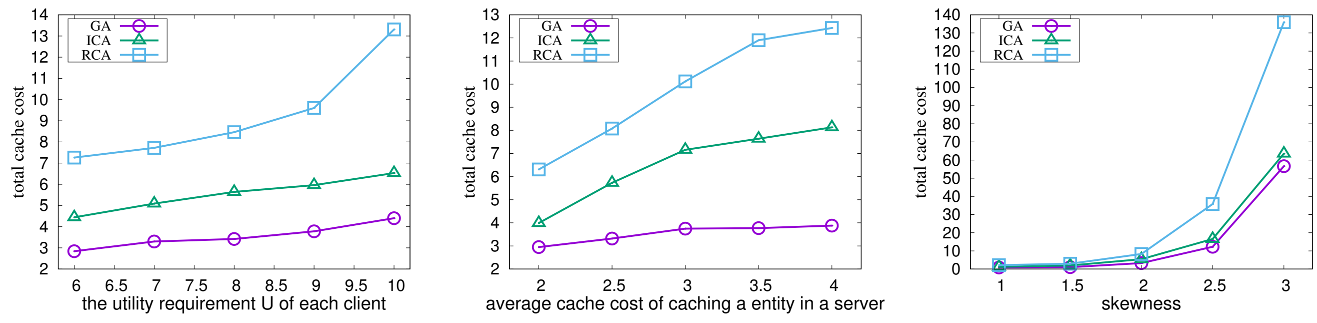
In Figs. 2(a), 3(a), 4(a), and 5(a), when the utility requirement increases, we find all of GA, ICA, and RCA have to use more cache cost to satisfy the utility requirement of every client. This is reasonable, since a larger utility requirement of a client cannot be satisfied by caching fewer service entities.

It should be noted that the gap between GA and ICA in the delay-function-as-utility setting (e.g., Figs. 3(a)



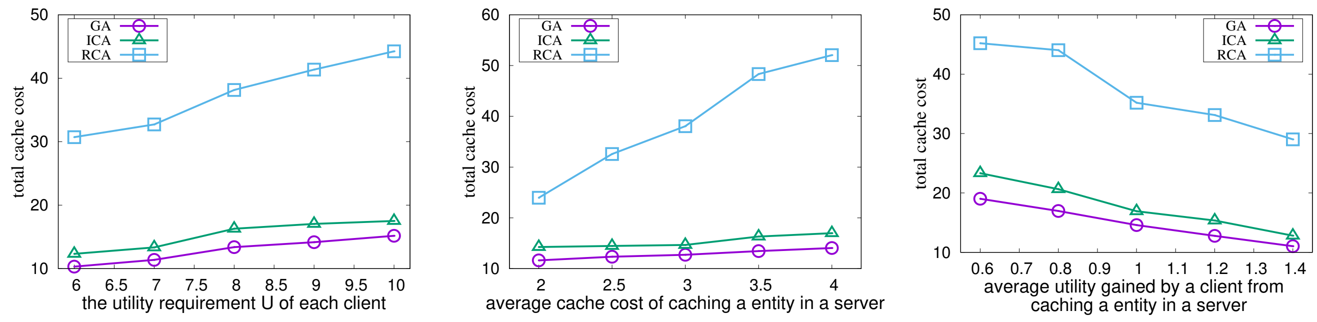
(a) Impact of utility requirement (b) Impact of average cache cost (c) Impact of average utility

Fig. 2: Evaluation results under the bandwdith-as-utility setting using Beijing trace.



(a) Impact of utility requirement (b) Impact of average cache cost (c) Impact of skewness

Fig. 3: Evaluation results under the delay-function-as-utility setting using Beijing trace.



(a) Impact of utility requirement (b) Impact of average cache cost (c) Impact of average utility

Fig. 4: Evaluation results under the bandwdith-as-utility setting using New York trace.

and 5(a)) is larger than that in the bandwidth-as-utility setting (e.g., Figs. 2(a) and 4(a)). For example, the cache cost of GA is 64.0% of that of ICA on average in Fig. 3(a); the cache cost of GA is 87.1% of that of ICA on average in Fig. 2(a). The main reason is that the distribution of the utilities $u(s, c)$'s in the delay-function-as-utility setting is not a uniform distribution, making GA have more opportunities to optimize the cache cost.

In Figs. 2(b), 3(b), 4(b), and 5(b), when the average cache cost increases, we find all of GA, ICA, and RCA have to use more cache cost to satisfy the utility requirement of every client. In Figs. 2(c), and 4(c), when average $u(s, c)$ increases, the cache cost of every algorithm decreases. In Figs. 3(c), and 5(c), when the skewness in the delay-function-as-utility setting increases, then $u(s, c)$ becomes smaller on average, which in turn makes each algorithm pay more cache cost to satisfy the utility requirement of every client.

6.2.2 *Impact of Number of Servers and Number of Clients*

Fig. 6 shows the comparison results under the synthetical trace in which the default number of edge servers is 400 and the default number of clients is 5,000. We see that, even in this large scale, GA achieves a much smaller cache cost than ICA and RCA. Most of the findings from the previous figures still hold here.

We would like to highlight here that when the number of edge severs increases, the gap between GA and ICA increases. For example, GA achieves a cache cost which is 72.5%, 71.4%, 66.8%, 63.9%, and 63.0% of that achieved by ICA when the number of edge servers is 200, 300, 400, 500, and 600, respectively. This is because more edge servers bring more opportunities to GA for minimizing the cache cost. This phenomenon is also observed when comparing Fig. 6(c) with Figs. 3(c) and 5(c), in which the advantage of GA over ICA increases when the scale of

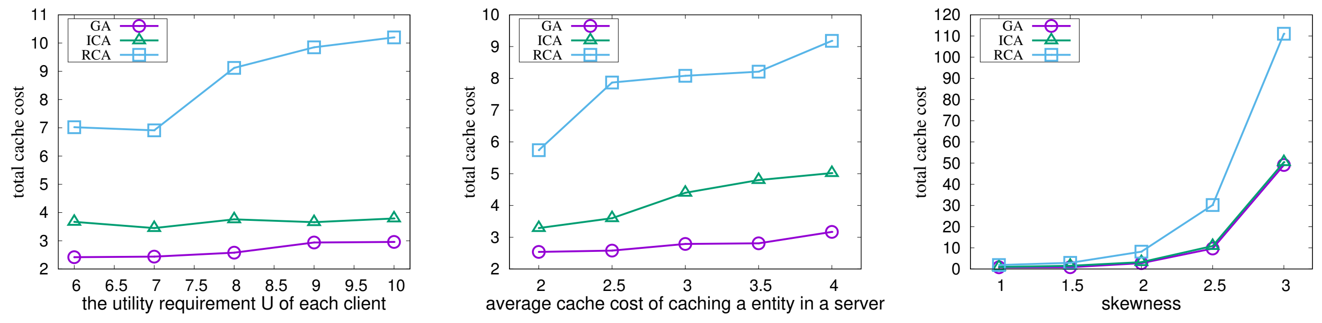

(a) Impact of utility requirement (b) Impact of average cache cost (c) Impact of skewness

Fig. 5: Evaluation results under the delay-function-as-utility setting using New York trace.

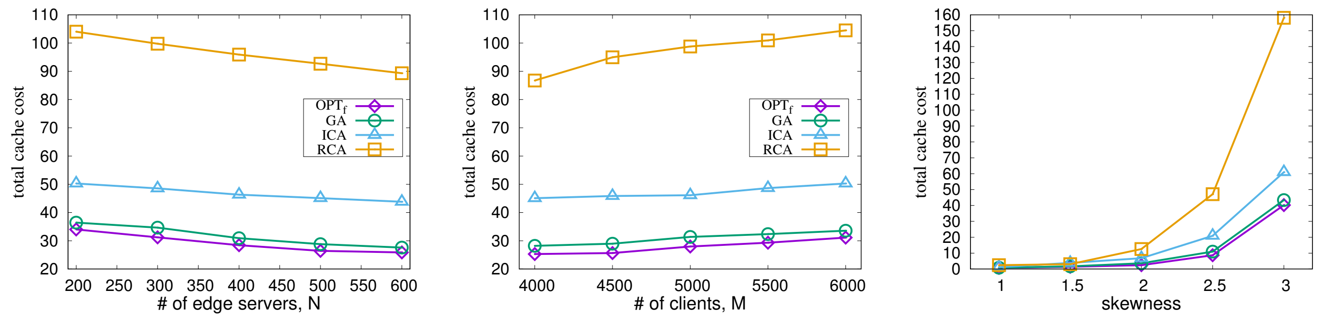

(a) Impact of the number of servers (b) Impact of the number of clients (c) Impact of skewness

Fig. 6: Evaluation results under the delay-function-as-utility setting using Synthetical trace.

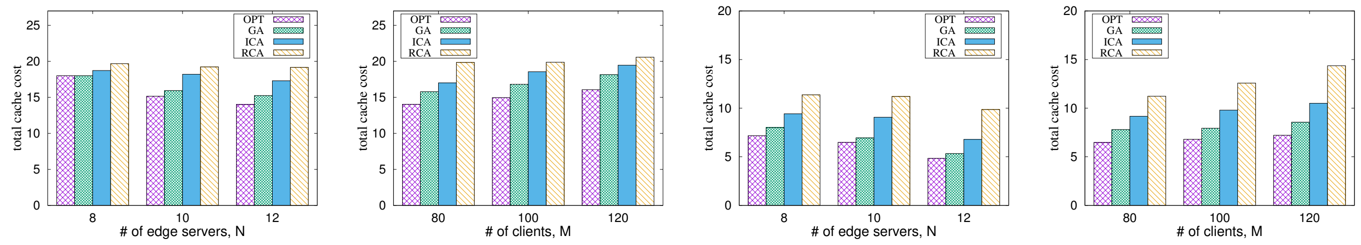

(a) Impact of the number of servers under the bandwidth-as-utility setting (b) Impact of the number of clients under the bandwidth-as-utility setting (c) Impact of the number of servers under the delay-function-as-utility setting (d) Impact of the number of clients under the delay-function-as-utility setting

Fig. 7: Comparison results of GA versus OPT using Synthetical trace.

the simulation increases (i.e., both of the number of edge servers and clients increases).

6.2.3 Approximation Ratio

As we mentioned in Section 5.2, due to the relaxation, the total cache cost of the optimal solution OPT_f to pfUtilitySEC is no more than that of the optimal solution OPT to UtilitySEC. OPT_f can be easily obtained by solving a linear programming problem pfUtilitySEC. Therefore, Fig. 6 also shows the performance of OPT$_f$. Throughout this set of simulations, the total cache cost achieved by GA is at most 149.0% (113.8% on average) as large as that achieved by OPT$_f$. These results suggest that, even compared with the optimal solution to the primal fractional problem, GA still achieves a near-optimal performance on average.

We are also interested in comparing GA with OPT. Since it is impractical to run the brute-force OPT in general, we evaluate the performance of GA, ICA, RCA, and OPT under a smaller setting, in which the default number of edge servers is 10 and the default number of clients is 100. Fig. 7 shows the comparison results under two types of utility settings. In general, the gap between GA and OPT is smaller in the bandwidth-as-utility setting than that in the delay-function-as-utility setting. Specifically, throughout this set of simulations, the total cache cost achieved by GA is at most 120.6% (111.4% on average) as large as that achieved by OPT, while the total cache cost achieved by ICA and RCA is at most 145.4% and 203% (129.7% and 155.7% on average), respectively, as large as that achieved by OPT. These results also imply that the approximation ratio in the theoretical analysis is not tight, and finding the tight ratio is left as part of future work.

7 Related Work

Application offloading is a relatively common topic. To improve energy efficiency of mobile devices, some works proposed offloading mobile workloads to nearby edges or remote clouds [6, 8, 9, 13, 15, 33]. For example, Cuervo et al. [6] proposed MAUI that enables automated method-level code offloading; Kosta et al. [13] improved method-level migration by parallelizing method execution using multiple VM images; Gordon et al. [8] designed thread-level migration techniques based on distributed shared memory; dynamic offloading with completion deadline constraint to reduce energy consumption was studied in [9], etc.

Some other works [5, 7, 25, 26, 29, 31, 37] focused on minimizing the completion time of mobile workloads. For example, Wang et al. [25] considered minimizing job latency from the perspective of edge cloud resource provisioning; Xu et al. [29] jointly considered caching and offloading decisions; Wang et al. [26] studied a similar service entity placement problem that resembles the uncapacitated facility location problem [24]; Gao et al. [7] considered both edge workload and access network congestion when placing service entities. Simultaneously optimizing offloading for multiple users was studied in [5, 31]. The virtual network functions placement for service-customized 5G networks was studied in [37]. Dispatching crowdsensing tasks was studied in [27, 28].

Dispatching multiple jobs to multiple edges was considered in [11, 21–23]. Tong et al. [23] proposed the hierarchical edge architecture. Tan et al. [22] proposed to greedily dispatch and schedule jobs using the Highest Residual Density First rule. Sundar and Liang [21] investigated the problem of dispatching dependent tasks to multiple edges with deadline constraints, so as to minimize application execution cost. Jia et al. [11] studied the load balancing between multiple edge clouds. Grouping jobs with complementary demands to optimize resource utilization was studied in [32].

Optimizing edge for mobile augmented reality (MAR) applications receives intensive research interests recently. Liu et al. [16] focused on edge server assignment and frame resolution selection to minimize MAR service latency. VideoStorm [35] leveraged the resource-quality tradeoff and latency-tolerance of partial video analysis requests to accelerate video analysis. Chameleon [12] utilized temporal persistence of top-k configurations and spatial similarities to minimize the resource consumption for video analysis. Focus [10] used top-k index and object clustering to achieve low-latency and low-cost video analysis. These studies explore domain-specific knowledge to optimize edge video analysis.

In brief, none of existing studies investigate the utility-based service entity placement problem, in which we can generalize the proposed problem and algorithm to many existing problems by modifying the 'utility'. We provide a non-trivial NP-complete result and design an efficient approximation algorithm.

8 Conclusion

For edge service providers, service entity provisioning is an important problem. In this paper, we study this problem from the utility perspective. We leverage 'utility' to generalize the proposed UtilitySEC problem to many existing problems by modifying the 'utility'. We theoretically show that the decision version of UtilitySEC is NP-complete. We present GA, an approximation algorithm that greedily caches a service entity in an edge server that has the smallest cost-effectiveness. We evaluate GA with both real-world data traces and large-scale simulations, and observe that GA performs close to the optimal algorithm and generally outperforms the other baseline algorithms.

Acknowledgments

This work was supported in part by National Key R&D Program of China (2017YFB1001801, 2016YFC0800803), NSFC (61872175), Jiangsu NSF (BK20181252), CCF-Tencent Open Fund, and Collaborative Innovation Center of Novel Software Technology and Industrialization. Jidong Ge is the corresponding author.

References

[1] "Boeing 787," https://tinyurl.com/ya5tmhkk, accessed on Mar 2018.

[2] "Edge servers in Alibaba," https://www.aliyun.com/, accessed on Mar 2018.

[3] "Open Edge Computing," http://openedgecomputing.org/.

[4] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan, "Mapping the expansion of google's serving infrastructure," in *Proc. SIGCOMM IMC 2013*, pp. 313–326.

[5] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, October 2016.

[6] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "MAUI: making smartphones last longer with code offload," in *Proc. ACM MobiSys 2010*, pp. 49–62.

[7] B. Gao, Z. Zhou, F. Liu, and F. Xu, "Winning at the starting line: Joint network selection and service placement for mobile edge computing," in *Proc. IEEE INFOCOM 2019*, pp. 1–9.

[8] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, "COMET: code offload by migrating execution transparently," in *Proc. USENIX ODSI 2012*, pp. 93–106.

[9] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *Proc. IEEE INFOCOM 2016*, pp. 1–9.

[10] K. Hsieh, G. Ananthanarayanan, P. Bodik, S. Venkataraman, P. Bahl, M. Philipose, P. B. Gibbons, and O. Mutlu, "Focus: Querying large video datasets with low latency and low cost," in *Proc. USENIX OSDI 2018)*, pp. 269–286.

[11] M. Jia, W. Liang, Z. Xu, and M. Huang, "Cloudlet load balancing in wireless metropolitan area networks," in *Proc. IEEE INFOCOM 2016*, pp. 1–9.

[12] J. Jiang, G. Ananthanarayanan, P. Bodk, S. Sen, and I. Stoica, "Chameleon: Scalable adaptation of video analytics," in *Proc. ACM SIGCOMM 2018*, pp. 1–14.

[13] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *Proc. IEEE INFOCOM 2012*, pp. 945–953.

[14] F. Liu, P. Shu, H. Jin, L. Ding, J. Yu, D. Niu, and B. Li, "Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications," *IEEE Wireless Communications*, vol. 20, no. 3, pp. 14–22, June 2013.

[15] F. Liu, P. Shu, and J. C. S. Lui, "AppATP: An energy conserving adaptive mobile-cloud transmission protocol," *IEEE Transactions on Computers*, vol. 64, no. 11, pp. 3051–3063, Nov 2015.

[16] Q. Liu, S. Huang, J. Opadere, and T. Han, "An edge network orchestrator for mobile augmented reality," in *Proc. IEEE INFOCOM 2018*, pp. 756–764.

[17] Y. Niu, F. Liu, and Z. Li, "Load balancing across microservices," in *Proc. IEEE INFOCOM 2018*, pp. 198–206.

[18] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[19] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[20] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[21] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *Proc. IEEE INFOCOM 2018*, pp. 37–45.

[22] H. Tan, Z. Han, X.-Y. Li, and F. C. Lau, "Online job dispatching and scheduling in edge-clouds," in *Proc. IEEE INFOCOM 2017*, 2017, pp. 1–9.

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

[24] V. Vazirani, *Approximation algorithms*. Springer, 2004.

[25] C. Wang, S. Zhang, H. Zhang, Z. Qian, and S. Lu, "Edge cloud capacity allocation for low delay computing on mobile devices," in *Proc. IEEE ISPA 2017*, pp. 290–297.

[26] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser, "Service entity placement for social virtual reality applications in edge computing," in *Proc. IEEE INFOCOM 2018*, pp. 468–476.

[27] M. Xiao, J. Wu, and L. Huang, "Community-aware opportunistic routing in mobile social networks," *IEEE Transactions on Computers*, vol. 63, no. 7, pp. 1682–1695, July 2014.

[28] M. Xiao, J. Wu, L. Huang, R. Cheng, and Y. Wang, "Online task assignment for crowdsensing in predictable mobile social networks," *IEEE Transactions on Mobile Computing*, vol. 16, no. 8, pp. 2306–2320, Aug 2017.

[29] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE INFOCOM 2018*, pp. 207–215.

[30] D. Yang, X. Fang, and G. Xue, "ESPN: Efficient server placement in probabilistic networks with budget constraint," in *Proc. IEEE INFOCOM 2011*, pp. 1269–1277.

[31] L. Yang, J. Cao, H. Cheng, and Y. Ji, "Multi-user computation partitioning for latency sensitive mobile cloud applications," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2253–2266, 2015.

[32] X. Yi, F. Liu, D. Niu, H. Jin, and J. C. S. Lui, "Cocoa: Dynamic container-based group buying strategies for cloud computing," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 2, no. 2, pp. 8:1–8:31, Feb. 2017.

[33] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *arXiv preprint arXiv:1605.08518*, 2016.

[34] R. Yu, G. Xue, and X. Zhang, "Application provisioning in fog computing-enabled internet-of-things: A network perspective," in *Proc. IEEE INFOCOM 2018*, pp. 783–791.

[35] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *Proc. USENIX NSDI 2017*, pp. 377–392.

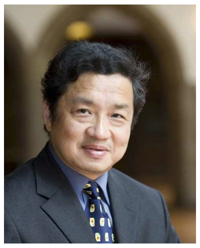
[36] L. Zhang and X. Tang, "Client assignment for improving interactivity in distributed interactive applications," in *Proc. IEEE INFOCOM 2011*, pp. 3227–3235.

[37] Q. Zhang, F. Liu, and C. Zeng, "Adaptive interference-aware vnf placement for service-customized 5g network slices," in *Proc. IEEE INFOCOM 2019*, pp. 1–9.

Yu Liang is a PhD candiate in Nanjing University. She received her MS degree from Nanjing University in 2011. She was a senior software engineer in Trend Micro China Development Center between 2011 and 2017. Her research interests include resource allocation in cloud and edge computing. Her publications include those appeared in Elsevier Computer Communications, IEEE CDCS, and IEEE Globecom.

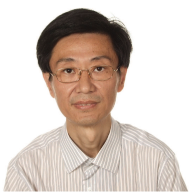
Jidong Ge is an Associate Professor at Software Institute, Nanjing University. He received his PhD degree in Computer Science from Nanjing University in 2007. His current research interests include cloud computing, workflow scheduling, software engineering, workflow modeling, process mining. His research results have been published in more than 60 papers in international journals and conference proceedings including IEEE TSC, JASE, COMNET, JPDC FGCS, JSS, Inf. Sci., JNCA, JSEP, ESA, ICSE, IWQoS etc.

Sheng Zhang is an assistant professor in the Department of Computer Science and Technology, Nanjing University. He is also a member of the State Key Lab. for Novel Software Technology. He received his BS and PhD degrees from Nanjing University in 2008 and 2014, respectively. His research interests include cloud and edge computing, network function virtualization and mobile computing. He has published more than 50 papers, including those appeared in *IEEE TMC*, *IEEE TPDS*, *IEEE TC*, *Elsevier COMNET*, *ACM MobiHoc*, *IEEE ICDCS*, *IEEE INFOCOM*, *IEEE/ACM IWQoS*, and *ICPP*. He received the Best Paper Runner-Up Award from *IEEE MASS* 2012. He is a member of the IEEE.



Jie Wu (F'09) is the Director of the Center for Networked Computing and Laura H. Carnell professor at Temple University. He also serves as the Director of International Affairs at College of Science and Technology. He served as Chair of Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and Associate Vice Provost for International Affairs from the fall of 2015 to the summer of 2017. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Mobile Computing, IEEE Transactions on Service Computing, Journal of Parallel and Distributed Computing, and Journal of Computer Science and Technology. Dr. Wu was general co-chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, ACM MobiHoc 2014, ICPP 2016, and IEEE CNS 2016, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.

Ze Tang received the BS degree in the Software Institute, Nanjing University. He is currently working toward the MS degree in the Software Institute, Nanjing University, under the supervision of Prof. Jidong Ge. His research interests include cloud computing and NLP.



Bin Luo is a full Professor at the Software Institute, Nanjing University. His main research interests include cloud computing, computer network, workflow scheduling, software engineering. His research results have been published in more than 60 papers in international journals and conference proceedings including IEEE TSC, ACM TOIST, FGCS, JSS, Inf. Sci., ESA etc. He is leading the institute of applied software engineering at Nanjing University.