Data Placement Strategies for Data-Intensive Computing over Edge Clouds

Xinliang Wei*, A B M Mohaimenur Rahman†, Yu Wang*
*Department of Computer and Information Sciences, Temple University, Philadelphia, PA, USA
†Department of Computer Science, University of North Carolina at Charlotte, Charlotte, NC, USA

***Abstract*—Edge computing has become an increasingly popular computing paradigm. Deploying edge clouds allows performing data-intensive computing at the edge of the network instead of a remote cloud to reduce data access latency and improve data processing efficiency. One of the key challenges in data-intensive edge computing is how to effectively place the data at the edge clouds such that the access latency to the data is minimized. In this paper, we study such a data placement problem in edge computing where different data items have diverse popularity. We first propose a data popularity based placement method when the data requests are unknown. It maps both data items and edge servers to a virtual plane and places data based on its virtual coordinate in the plane. We consider data popularity during both the mapping of data items to the plane and making the placement decision. We further propose an optimization-based placement strategy for the case when the data requests are known. By formulating an integer programming problem, our proposed solution aims to find the optimal placement decision. Simulation results show that both proposed strategies efficiently reduce the average latency of data access.**

I. Introduction

With the increasing amount of data generated by diverse applications and devices, especially the large amount of data collected by pervasive and mobile devices, data transmission has become the bottleneck of traditional cloud-based platforms. Sending all the data to the cloud for data processing or intelligent services is time-consuming and causes long response latency. Therefore, a recent trend is to process the data at the edge of the network near the users to shorten the response time, improve processing efficiency, and reduce network congestion. In addition, with the advancement of federated learning and artificial intelligence of things (AIoT), not only millions of data are generated from daily smart devices, such as smart light bulbs, cameras, various sensors, but also a large number of parameters of complex machine learning (ML) models have to be trained and exchanged by these AIoT devices. Classical cloud-based platforms have struggled to effectively communicate and process these data or models with sufficient privacy and secure protection. This has further accelerated the growth of this new computing paradigm - data-intensive edge computing [1]–[3].

As shown in Fig. 1, a typical edge computing environment consists of mobile users, edge servers, the edge network, and a remote cloud. Edge servers are geographically dispersed at the edge of the network near the mobile users, and own heterogeneous computing and storage capability [1]. Each edge server can provide data-intensive services (such as video analytics, augmented reality, distributed machine learning) for those mobile users in the specific nearby area by holding some data items or ML models and performing the computation based on these data and models. Hereafter, we use data to refer to both data and models as long as they are required to perform the service requested by the mobile users. When a mobile user requests data, its request is forwarded to the nearest edge server. If the edge server has the data, it can respond to the mobile user immediately with the data (as data d_1 in Fig. 1) or perform the corresponding computing service for the user if needed. Otherwise, the edge server has to retrieve the data from other edge servers (data d_2 or d_5) or even from the remote cloud (data d_3). Clearly, data placement is a critical issue in edge computing since the location of data affects the response latency of the requested service. If the data is stored at a nearby edge server, the service can be performed very quickly, while a request needed to access a remote cloud takes much longer to be performed. In addition, as shown in Fig. 1, multiple mobile users at different locations may request the same data (data d_2), and different data has diverse popularity (i.e., different number of requests from users). Therefore, in this paper, we study the data placement problem in edge computing with the consideration of data popularity from user requests.

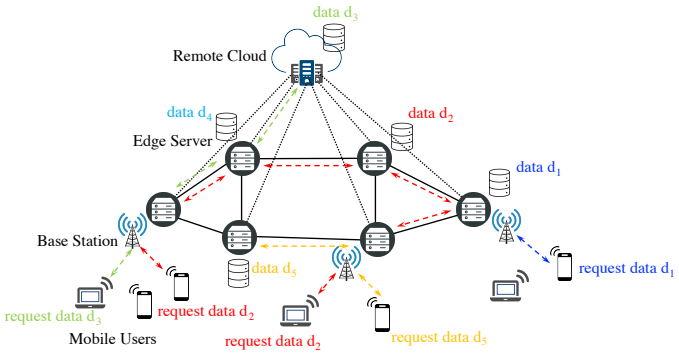


Fig. 1. An edge-cloud computing environment where data items are placed at edge servers or the remote cloud server.

Data placement has been well studied in distributed systems [4]. However, edge computing has its own characteristics [1]

The work is partially supported by the US National Science Foundation under Grant No. CCF-1908843 and CNS-2006604.

978-1-6654-4331-9/21/$31.00 ©2021 IEEE

(such as time-sensitive data, diverse edge resources, data/ML privacy concerns), thus data placement problem in edge computing has also drawn significant attentions from researchers recently [5]–[8]. Both [5] and [6] have studied the data placement strategy for workflows in edge computing, where workflow's dependency, reliability, and user cooperation are considered. Both use intelligent swarm optimization methods to solve the optimization problem. Li *et al.* [7] investigated a joint optimization of data placement and task scheduling to reduce the computation delay and response time. To solve the formulated data placement optimization, a tabu search algorithm designed for the knapsack problem is used. Most of these optimization-based methods are usually suffering from poor stability and high overheads. Breitbach *et al.* [8] have also studied both data placement and task placement by considering multiple context dimensions. For its data placement part, the proposed data management scheme adopts a context-aware replication, where the parameters of the replication strategy are tuned based on context information. Most recently, Xie *et al.* [9], [10] proposed a novel virtual space-based method, which maps both switches and data indexes/items into a virtual space and places data based on the virtual distance in the space. Their method can enable efficient retrieval via greedy forwarding. However, none of them consider data popularity when placing data on edge servers in edge computing.

In this paper, we investigate the data placement strategy in edge computing with the aim of reducing the average access latency of data. We consider two scenarios: (1) data requests are unknown, but data popularity (defined as request frequencies of data items) are known; (2) data requests are known. For the first scenario, inspired by [9], [10], we adopt a virtual-space based placement method, but take into consideration of data popularity when we generate the coordinates of data items. Based on an observation that in a dense network, the node in center region has smaller shortest path to other areas compared with nodes in the surrounding regions, we carefully design our mapping strategy so that a popular data item is placed closer to the network center in the virtual plane. Then the placement of data is purely based on the distance between data item and edge server in the virtual plane. To address the storage limits at servers, we also consider how to offload data items to other servers by processing the mapping in an order based on data popularity. For the second scenario, with the detailed data requests, we formulated a data placement optimization problem as an integer linear programming (ILP) problem. Such problem can be solved by existing ILP solver or by a simple greedy heuristic. Finally, we conduct extensive simulations with both synthetic data and real world tracing data. Simulation results show that our proposed data popularity based strategies can achieve better performance compared to existing solutions [9], [10] for the first scenario. For the second scenario, the optimization-based solution can find much better solution than the popularity-based methods.

In summary, the contributions of this paper are three-folds.

- To our best knowledge, our proposed data popularity based data placement strategy is the first virtual-space based method to consider data popularity in data placement in edge computing. Our proposed method maps more popular data closer to the network center and thus it is placed to a nearby server, which shortens the shortest paths during the data retrieval process and reduces the overall response latency.
- We also propose an optimization-based data placement strategy which can find optimal placement decision given that the data requests are known.
- We have conducted extensive simulations to verify the efficiency and effectiveness of the proposed data placement strategies. It confirms the advancement of taking data popularity into consideration in our design.

The rest of this paper is organized as follows. Section II introduces the system model used. Section III and Section IV present our proposed placement schemes based on data popularity and optimization, respectively. Evaluations of the proposed methods are provided in Section V. Finally, Section VI concludes the paper with possible future directions.

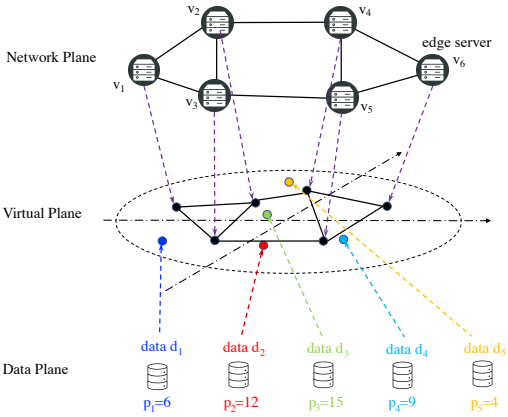


Fig. 2. **Virtual space based approach**: edge servers and data items are mapping to a virtual plane and associated with virtual coordinates.

II. System Model

We consider a network formed by edge clouds, as shown in the top layer of Fig. 2 (i.e., network plane) and denoted by $G(V, E)$. Here, $V = \{v_1, v_2, \cdots, v_N\}$ is a set of N edge servers and $E = \{e_1, e_2, \cdots, e_M\}$ is a set of M direct links between edge servers. Each edge server v_i has a maximal capacity $c_i = c(v_i)$ of its storage. For a direct link between edge server v_i and v_j denoted as $e_{i,j}$, we use $|e_{i,j}|$ and $\tau_{i,j} = \tau(e_{i,j})$ to represent its physical distance and propagation delay, respectively. We assume that the propagation delay is linearly proportional to the physical distance. For a given pair of edge servers v_i and v_j, we use $l_{i,j} = l(v_i, v_j)$ to represent the shortest path length in term of total delay from v_i to v_j in G. We then have a delay matrix $L = \{l_{i,j}\}$ which holds total delay of all the shortest paths in the edge network.

Assume that there are a set of data items $D = \{d_1, d_2, \cdots, d_W\}$ in the data-intensive edge environment, which could be multiple types of data, such as text, image, video, IoT sensing or scientific data. Each data d_i has a specific

TABLE I
SUMMARY OF NOTATIONS.

Symbol	Notation or Definition
V, E, D, R	the set of edge servers, links, data items and requests
N, M, W, U	the number of edge servers, links, data and requests
v_i, d_l, r_k	a edge server, a data item, and a data request
$e_{i,j}, \tau_{i,j}$	a direct link between v_i and v_j, its propagation delay
c_i	the maximal storage capacity of server v_i
$l_{i,j}$	the shortest path length from server v_i to v_j
s_i, p_i	the size and popularity of data d_i
$v(r_i), d(r_i)$	the arriving server and requested data of r_i
$f(d_i) = v_j$	a data place mapping f places d_i to v_j
L	the network delay matrix, $L = \{l_{i,j}\}$
B	the scalar product matrix, $B = \frac{1}{2} J L^{(2)} J$
λ_1, λ_2, v_1, v_2	two largest eigenvalues and their eigenvectors of B
Q	the coordinate matrix of servers
q_{max}	the largest absolute value of elements in Q
p_{max}	the maximal data popularity
$H(d_l), h(d_l)$	the hash value and its 4-byte binary value of d_l
$r(d_l), \theta(d_l)$	the Polar coordinate of d_l
$x(d_l), y(d_l)$	the Cartesian coordinate of d_l
$x_{l,i}$ ($x^t_{l,i}$)	the data placement decision (at time t)), $x_{l,i} \in \{0, 1\}$
ζ_l	the downloading cost from the cloud of d_l
μ_l	the placement cost of d_l

size $s_i = s(d_i)$ and popularity $p_i = p(d_i)$ (which will be explained in Section III-A). We assume that there are also a sequence of data requests $R = \{r_1, r_2, \cdots, r_U\}$, each of which arrives at a server $v(r_i)$ and asks for one data item $d(r_i)$. Note that if a request includes multiple data items we can treat it as multiple requests.

Then, the data placement problem aims to find an edge server v_j for each of data item d_i to hold it. Such a problem can be represented as finding a mapping f from D to V, where $f(d_i) = v_j$. The goal of data placement problem is to find a mapping to minimize the average access cost (or delay) of all requests to stored data items in edge network G and also satisfy the storage constraint (i.e., the storage of data items does not exceed the server's storage capacity).

Table I summaries the notations used in our system model and proposed methods.

III. Data Popularity based Data Placement

In this section, we first introduce data popularity and then devise a data placement strategy based on it.

A. Data Popularity

Data popularity measures how much a given piece of data is requested by the users. This gives an indication of the importance of that data. Taking popularity into account allows to place the data or their replicas better to improve access efficiency in any distributed systems [4]. This is also true for the data placement problem in data-intensive edge computing. Obviously, placing more popular data items at the edge server with shorter delay within the network can significantly reduce the data access cost during data retrievals, since popular data are repeatedly requested by various users from all edge servers. Therefore, in this section, we introduce data popularity to assist the data placement strategy in edge computing.

Although data popularity has been widely used in distributed systems, to our best knowledge, most of the existing data placement strategies for edge computing do not consider data popularity. The only exception is [7], where the authors considered data popularity as a part of their estimation of value of data block in their formulated placement problem. The placement problem is then formulated as a complex combinatorial optimization problem solved by a tabu search algorithm. Different from their solution, we use data popularity in the virtual space mapping where data items are mapped to a virtual space based on their popularity, and then the placement decision is made based on the coordinates in the virtual space.

Data popularity can be assessed differently depending on the application and generally based on three factors: the number of accesses (or requests), the lifetime, and the request distribution over time/space. In this paper, we simply use the number of requests as the data popularity. However, it is not difficult to extend our definition to include other two factors (or even other popularity measurements). For each data item d_i, we assume that its popularity $p_i = p(d_i)$ describes its number of access requests over time. We consider two ways to determine the popularity. In a static setting, we assume data popularity for each data is static over a long period of time and is known to the system. In a dynamic setting, we assume that data popularity varies over time and can be dynamically adjusted based on a specific time window. In both cases, larger data popularity means the data item is more frequently accessed by mobile users. Obviously, the locations of popular data items are at the roots of the overall data placement problem.

B. Data Popularity based Data Placement

Our data popularity-based data placement strategy adopts a virtual-space approach similar to [9], [10]. It maintains a virtual two-dimensional circular plane and maps all edge servers and data items to such a plane, as shown in Fig. 2. How to perform the mappings is critical in our design. We first construct the virtual coordinates of edge servers in the plane based on the delay distribution among them in the network, then map the data items onto the plane by calculating their virtual coordinates based on their data popularity. When mapping the data items onto the plane, we try to spread them out while making sure that the more popular data is closer to the center. The intuition behind this design is the shortest paths to all other servers are shorter at the center area. Finally, we place data items to the edge server with the closest distance in the virtual plane. Note that [9], [10] do not consider the data popularity of data items in their virtual-space-based placement methods. Next, we present these specific steps in detail.

1) Coordinate Construction: To obtain the *coordinates of edge servers*, we basically adopt the M-position algorithm proposed by [9], [10]. The basic idea of the M-position algorithm is to calculate the coordinates matrix of edge servers based on the eigenvalue decomposition technique. The coordinates matrix is represented as Q which is a $2 \times N$ matrix where N is the number of edge servers. The input mainly consists of the

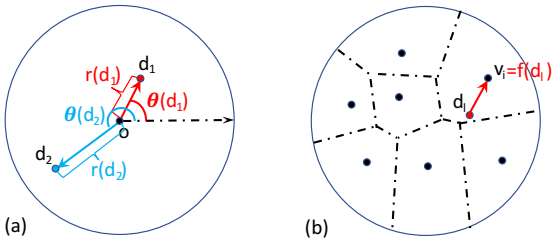


Fig. 3. Data popularity based placement: (a) virtual coordinates of two data items in the virtual plane with Polar coordinates; (b) data placement of d_l to the nearest server (its Voronoi cell owner) in the virtual plane.

network delay matrix $L = \{l_{i,j}\}$. The detail steps of mapping edge servers to the virtual plane are listed as follows.

a. We first calculate the delay matrix L based on the network delay $\tau_{i,j}$ on edges and then compute the square of L, i.e., $L^{(2)} = \{l_{i,j}^2\}$.
b. Next, we generate the scalar product matrix $B = \frac{1}{2}JL^{(2)}J$ according to $L^{(2)}$ and the matrix $J = 1 - \frac{1}{N}A$, where A is a matrix of ones with $N \times N$ size.
c. Then, we find the two largest eigenvalues λ_1, λ_2 as well as their corresponding eigenvectors υ_1, υ_2 of matrix B. Let Υ and Λ be the matrix of two eigenvectors and the diagonal matrix of two eigenvalues, respectively. We can then compute the coordinate matrix of servers $Q = \Upsilon\Lambda^{1/2}$.
d. At last, to place edge servers within a circular region with unit radius in the virtual space, we normalize Q to $\frac{1}{\sqrt{2}q_{max}}Q$, where q_{max} is the largest absolute value of elements in Q.

By doing so, we can construct the coordinate of edge servers and map them within a circular region in the virtual plane. The Euclidean distance between two servers in the virtual plane is proportional to their network distance (i.e., delay).

To generate the *coordinates of data*, we leverage a hash function and the Polar coordinate system (where a point can be represented by a distance r and angle θ, as shown in Fig. 3(a)). Firstly, we try to spread data over the virtual plane in order to balance the load of edge servers. Secondly, we take data popularity into consideration and place the more popular data to the location which has a smaller total delay to other edge servers. Lastly, the coordinates of data should be deterministic. Namely, whenever we provide the same data, the coordinate of data should be the same so that the retrieval process can get the same location of data when the task requests the same data. The specific steps for calculating the data coordinate of data item d_l are as follows.

a. Calculate the distance $r(d_l)$ of data d_l based on p_l via $r(d_l) = 1 - p_l/p_{max}$, where p_{max} is the maximal data popularity among all data. In the example shown in Fig. 3(a), d_1 is more popular than d_2's, thus its distance to the center is shorter.
b. Calculate the angle $\theta(d_l)$ of d_l by using the hash value of data index/ID. We make use of SHA-256 hash function to generate the hash value $H(d_l)$, and convert the last 8 bytes of the hash value $H(d_l)$ to a 4-byte binary value $h(d_l)$. We further normalize $h(d_l)$ to the range $[0, 2\pi]$, e.g., $\theta(d_l) = 2\pi \times h(d_l)/(2^{32} - 1)$. In this way, data will be spread in different directions. Even those data with the same data popularity will be mapped in the different coordinates.
c. Convert the Polar coordinate $(r(d_l), \theta(d_l))$ to the Cartesian coordinate, i.e., $x(d_l) = r(d_l)\cos\theta(d_l)$ and $y(d_l) = r(d_l)\sin\theta(d_l)$.

Algorithm 1 Popularity based Data Placement (DP-1)

Input: The list of data items d_l and edge servers v_i and their parameters (such as data size s_l, data popularity p_l, storage capacity c_i), the edge network $G(V, E)$ and network delay $\tau_{i,j}$ on its edges.
Output: The data placement decision $f(d_l)$.

Coordinate Construction for Edge Servers:

1: Calculate the network delay matrix $L = \{l_{i,j}\}$ with an all-pairs shortest path algorithm.
2: Prepare $L^{(2)} = \{l_{i,j}^2\}$ and $J = 1 - \frac{1}{N}A$.
3: Calculate the scalar product matrix $B = \frac{1}{2}JL^{(2)}J$.
4: Prepare Υ and Λ by finding λ_1, λ_2, υ_1, υ_2 of matrix B.
5: Compute the coordinate matrix of servers $Q = \Upsilon\Lambda^{1/2}$.
6: Normalize Q via $Q = \frac{1}{\sqrt{2}q_{max}}Q$, then coordinate $(x(v_i), y(v_i))$ of server v_i can be obtained from Q.

Coordinate Construction for Data:

1: **for** each data d_l **do**
2: Calculate its distance $r(d_l) = 1 - p_l/p_{max}$.
3: Compute its angle $\theta(d_l) = 2\pi \times h(d_l)/(2^{32} - 1)$.
4: Convert the Polar coordinate to the Cartesian one: $(x(d_l), y(d_l)) = (r(d_l)\cos\theta(d_l), r(d_l)\sin\theta(d_l))$.
5: **end for**

Data Placement Decision:

1: **for** each data d_l **do**
2: Find the nearest edge server with minimal Euclidean distance in the virtual plane: $f(d_l) = \arg\min_{v_i} ||(x(v_i), y(v_i)), (x(d_l), y(d_l))||$.
3: **end for**

return the data placement decision $f(d_l)$.

2) Data Placement: After we construct the coordinate of edge servers and data items, it is straightforward to place data to edge servers. Our goal is to place the data to the nearest edge server with minimal Euclidean distance in the virtual plane, i.e.,

$$f(d_l) = \arg\min_{v_i} ||(x(v_i), y(v_i)), (x(d_l), y(d_l))|| = \arg\min_{v_i} \sqrt{(x(v_i) - x(d_l))^2 + (y(v_i) - y(d_l))^2}.$$

In other words, we can first form a Voronoi diagram based on the edge servers' coordinates in the virtual plane, as shown in Fig. 3(b). If a data item is mapped within a Voronoi cell,

then it will be placed at the edge server owning that Voronoi cell. In this matter, the more popular data will be placed to the location near the center of the network, thus having a smaller average delay.

Overall, Algorithm 1 shows the detail of this data placement solution, which we call **Basic Data Placement based on Data Popularity** (DP-1). The construction of coordinates for all edge servers takes $O(N^3)$, which is dominated by the complexity of all-pairs shortest path and eigen decomposition of the matrix. The construction of coordinates for all data items can be done in $O(W)$. The data placement decision is made within $O(WN)$. Therefore, the total time complexity of DP-1 is $O(N^3 + WN)$. Recall that N and W are the number of edge servers and data items, respectively.

If an edge server has limited storage capacity, then one edge server may not hold all the data within its Voronoi cell. We need to offload some data to other servers. Therefore, we design a data popularity based offloading strategy. Basically, we first sort the data items based on their data popularity in descending order and then place each data item one by one following such order. When considering a data item d_l, we first find $v_i = f(d_l)$ based on the basic data placement (DP-1). If the current storage plus the data size of d_l does not exceed the maximal capacity $c(v_i)$ of this server, then d_l is placed in v_i. Otherwise, we offload this data to other edge servers by finding the nearest neighbor edge server v_j with available storage capacity enough for this data item. This process repeats for every data item, and thus all data are placed in appropriate edge servers while not violating the storage constraint. We call this placement strategy **Data Popularity based Data Placement** (DP-2). The total time complexity of DP-2 is $O(N^3 + WN + W \log W)$, where the additional $O(W \log W)$ is from ordering the data popularity.

3) Data Access: After data placement, data d_l can be accessed via the shortest path routing towards the placed server $f(d_l)$. In addition, virtual-space based solution can also enable a greedy forwarding strategy, where each server only need to know the coordinates of its neighboring servers. In such a method, the data request is greedily routed towards the coordinate $(x(d_l), y(d_l))$ of the data item in virtual plane. Although greedy forwarding may fail at a local minimum, randomized or Delaunay based [11]–[13] methods can be used to recover from the local minimum. This greedy forwarding method enjoys lower storage at servers and switches, as shown in [9], [10]. When offloading (as in DP-2) is used, an additional relay from the original target is needed for greedy forwarding.

IV. Optimization based Data Placement

While the proposed data popularity data placement can place popular data in the network center to shorten the average delay, the real delay of each data in the network is still far from optimal. In this section, with the assumption that data requests (i.e., a sequence of data requests $R = \{r_1, r_2, \cdots, r_U\}$) are known, we model the data placement as an optimization problem and further introduce an optimization-based data placement strategy.

A. Problem Formulation

We first introduce a binary indicator $x_{l,i}$ to store the data placement decision of d_l, i.e.,

$$x_{l,i} = \begin{cases} 1, & d_l \text{ is placed at server } v_i, \text{ i.e.,} f(d_l) = v_i, \\ 0, & \text{otherwise.} \end{cases}$$

Recall that data will be only placed in a single edge server, while one server can hold a couple of data but it cannot violate its maximal storage capacity. Therefore,

$$\sum_{i=1}^{N} x_{l,i} = 1, \text{ for all } d_l \text{ and } \sum_{l=1}^{W} x_{l,i} s_l \le c_i, \text{ for all } v_i.$$

The goal of data placement is to reduce the total accessing cost (or delay) for all requests R. Here, the total accessing delay can be defined as

$$\sum_{k=1}^{U} l_{v(r_k), f(d(r_k))} = \sum_{k=1}^{U} \sum_{i=1}^{N} x_{d(r_k),i} \cdot l_{v(r_k),i}.$$

Recall that $v(r_k)$ and $d(r_k)$ are the original requesting server and the requested data item for request r_k, while $l_{i,j}$ is the shortest delay from v_i to v_j. Therefore, the overall *data placement problem* is defined as the following optimization problem.

$$\min \quad \sum_{k=1}^{U} \sum_{i=1}^{N} x_{d(r_k),i} \cdot l_{v(r_k),i} \tag{1}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} x_{l,i} = 1, \qquad \forall l \tag{2}$$

$$\sum_{l=1}^{N} x_{l,i} s_l \le c_i, \qquad \forall i \tag{3}$$

$$x_{l,i} \in \{0, 1\}, \qquad \forall l, \forall i. \tag{4}$$

B. Solving the Optimization Problem

Obviously, the above optimization problem is an integer linear programming problem. Therefore, we can leverage linear programming techniques to solve it, such as branch and bound, and dynamic programming. In this paper, we make use of an optimal *ILP solver* [14] to determine the data placement decision $x_{l,i}$. We call this placement method **Optimization based Data Placement** (OPT).

In addition, we will also consider a simple greedy heuristic to solve the above optimization problem approximately. In this **Greedy Data Placement** (GRD), we also take data popularity into consideration. Specifically, we first sort all requested data items based on data popularity so that we can process the most popular data first. In each round, we only process one data item (say d_l) and make its placement decision greedily based on the access cost. Particularly, we calculate its access cost if we put d_l in each server with sufficient storage, and choose the server with the smallest access cost. This process stops until all requested data items are placed. The total time complexity of this method is $O(W \log W + N^3 + WN)$, where $O(W \log W)$, $O(N^3)$ and $O(WN)$ are for popularity ordering, all-pairs shortest path, and W rounds of greedy selection on access cost, respectively.

C. Further Discussion

So far, we only consider the accessing cost of data items from all requests. If the placement itself has a cost (such as downloading from the cloud or migrating from other servers), then the optimization-based method can also take that into consideration. This is similar when the data placement strategies need to be run periodically. Assume the data placement decision in time slot $t-1$ and t are $x_{l,i}^{t-1}$ and $x_{l,i}^{t}$, respectively. When we consider the data placement for t, we have to check whether the requested data is already stored at a server in the network at $t-1$.

In terms of placement cost, we consider two scenarios.

- A requested data item d_l has not been placed in the edge network yet. Thus $\sum_{i=1}^{N} x_{l,i}^{t-1} = 0$ and $\sum_{i=1}^{N} x_{l,i}^{t} = 1$ for each data d_l, $l \in X$. In this case, d_l needs to be downloaded from the remote cloud with a downloading cost ζ_l.
- The requested data d_l has been placed at v_i in the edge network, but it needs to be switched from v_i to v_j. In other words, $\sum_{i=1}^{N} x_{l,i}^{t-1} = 1$ and $\sum_{j=1}^{N} x_{l,j}^{t} = 1$, but $i \neq j$, $i, j \in N$.

Then, the placement cost of d_l can be defined as follows:

$$\mu_l = (1 - \sum_{i=1}^{N} x_{l,i}^{t-1}) \cdot \sum_{i=1}^{N} x_{l,i}^{t} \cdot \zeta_l + \sum_{i=1}^{N} \sum_{k \neq i, k=1}^{N} (x_{l,i}^{t} \cdot x_{l,k}^{t-1} \cdot l_{k,i}).$$

The objective of the optimization in each time slot becomes minimizing both accessing cost and placement cost, i.e.,

$$\min \sum_{l=1}^{X} \mu_l + \sum_{k=1}^{U} \sum_{i=1}^{N} x_{d(r_k),i}^{t} \cdot l_{v(r_k),i}. \quad (5)$$

All of the constraints (Equ. (2)-(4)) are still the same. Since this revised problem is also an integer linear programming problem, we can still use the ILP solver to determine the optimal placement decision $x_{l,i}^{t}$ in time t.

V. Performance Evaluation

We have evaluated the performance of the proposed algorithms via both synthetic and trace-driven simulations in a mobile edge computing simulator developed by our research group. We now first introduce the simulation setup, and then report the detailed simulation results.

A. Simulation Setup

For the synthetic simulation, to test our proposed data placement strategies, we randomly construct a network topology with 30 edge servers based on a binomial distribution. The delay (or cost) $\tau(e)$ of each link is uniformly drawn from $[1, 2]$. The maximal storage capacity of each server is randomly drawn from $[64, 128]$. We set the number of data items W to $1,000$. The size s_l of each data item d_l varies from $[1, 10]$, while its download cost ζ_l from remote cloud is linearly proportional to its size and ranges from $[1, 20]$. For data popularity p_l of each data, we randomly generate it from $[20, 50]$. Then a sequence of U data access requests are generated for a large time period, and the normalized frequency of data requests for each data item follows the generated data popularity. In our implementation, we allow each request to require multiple data items. In addition, to simulate the dynamic data popularity, we also calculate and update the data popularity of data item for a time window with size of 10. For each of the reported experiments, we perform multiple times and report the average results. The only performance metric is the *average accessing cost* of all requests (or the *average total cost*, which include both accessing and placement costs as defined in Section IV-C, of all requests when data placement is performed periodically). Here, the accessing cost is based on shortest path routing.

TABLE II
Parameters used in our simulations.

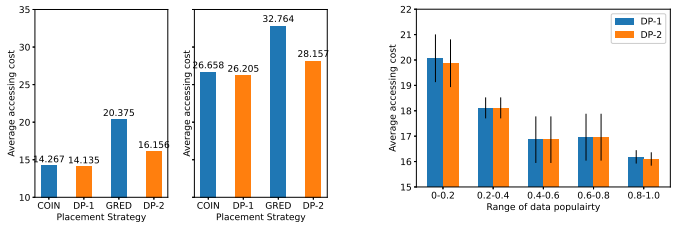
Parameter	Value
the number of edge servers	30
the delay of each link	[1, 2]
the maximal storage capacity of edge servers	[64, 128]
the maximal number of data items	1,000
the size of each data	[1, 10]
the download cost of each data	[1, 20]
the popularity of each data	[20, 50]
the length of time window	10
the length of time period	30 mins
the number of users	10
the number of data requests generated per user per min	[1, 4]
the number of data required by each request	[1, 6]

For the trace-driven simulation, we extract user mobility from the real mobility trace data. We use the CRAWDAD dataset kaist/wibro [15] developed by a Korean Team, which collected the CBR and VoIP traffic from the WiBro network in Seoul, Korea. We randomly sample a batch of data from this dataset and perform our simulation over a 30-minute period with user location updates every minute. Users will select the nearest edge server to perform the data request. The location of edge servers also comes from the trace data but fixed during the simulation. The number of user is 10 while the number of edge servers ranges from 10 to 30. Each user generates up to 4 data requests per minutes. So the maximal number of data requests is up to 40 per minutes. The number of data items required by each request ranges from 1 to 6. The other parameters are the same as the synthetic simulation. This set of simulations is only used for the evaluation of optimization based data placement methods proposed in Section IV.

Table II summaries the simulation parameters we used.

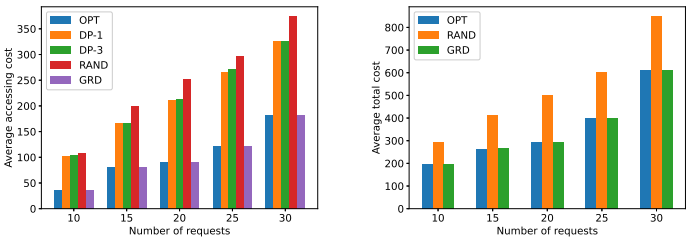
B. Results on Data Popularity based Data Placement

In this set of synthetic simulations, we consider that the data requests are unknown, but long-term data popularity is known. Therefore, the data placement is kind of static. We first compare our proposed data popularity based data placement strategies DP-1 and DP-2 against existing methods COIN [9] and GRED [10]. Both COIN and GRED also place data indexes or items using virtual space-based methods but do not consider data popularity. Compared with COIN, GRED considers load balance using the Centroidal Voronoi Tesselation technique when placing data to edge servers. DP-1 does not consider the server's capacity constraint, while



(a) against existing methods (b) cost vs popularity

Fig. 4. **Data popularity based methods:** (a) against existing placement methods [9], [10] where data requests are randomly generated at (left) any servers or (right) servers near the boundary. (b) average accessing cost of different data items with different data popularity.



(a) average accessing cost (b) average total cost

Fig. 5. **Optimization based methods:** Comparison of proposed solutions with different numbers of requests, when (a) only accessing cost is considered, or (b) both accessing cost and placement cost are considered.

DP-2 offloads data items to the nearest server if the original assigned server is full. In the experiments, we set the number of requests to 1000. Results are reported in Fig. 4(a). The left plot shows the result of the average accessing cost when data requests are generated at random edge servers. Clearly, if we do not consider storage capacity, our static data placement DP-1 has slightly better performance than COIN. However, if we consider the storage capacity, our proposed method DP-2 performs much better than GRED (which also considers the loads among servers). In the right plot, we consider a scenario where data requests randomly come from edge servers near the boundary of the network. Since these requests need a longer delay to reach other parts of the network, the accessing costs of all methods are higher than those in the left plot. Furthermore, we can observe that our proposed algorithms (DP-1 or DP-2) are much better than the existing methods (COIN or GRED) in this case. This is mainly because DP-1 and DP-2 consider data popularity and ensure that more popular data items are placed closer to the network center where the average accessing delay to boundary region is much smaller.

We further take a closer look at the accessing costs for different data items with various data popularity. Fig. 4(b) plots the average accessing costs of different groups of data items with a specific range of data popularity. Here, we group the data items into five groups based on their normalized popularity (ranging from 0 to 1) and then report the average accessing cost of each group. Clearly, the more popular data has a lower accessing cost. The overall trend in average costs decreases when data popularity increases. This confirms the advantage of taking data popularity into consideration for data placement.

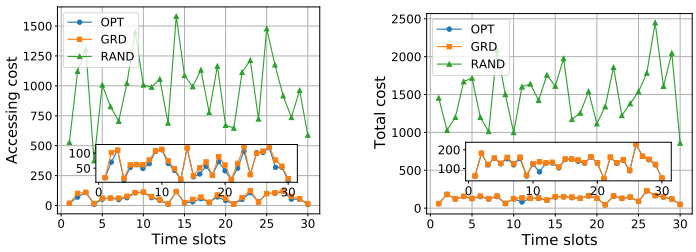
C. Results on Optimization based Data Placement

In this set of synthetic simulations, we assume that the requests R are known for data placement strategies, therefore, optimization-based methods can be used. In addition, we generate the 10-30 requests per time window (with a size of 10), so that we can also test our method based on dynamic data popularity (denoted as DP-3). Besides DP-1/DP-3, OPT, and GRD, we also implement a *random placement* (RAND) scheme, where data items are randomly placed at an edge server. We measure the average accessing costs of every method for serving all data requests and report them in Fig. 5(a). First, the average accessing cost of all methods increase as the number of data requests increases. This is reasonable, since serving more requests needs more access cost. Specifically, OPT and GRD can significantly reduce the average accessing cost, while the other three methods perform similarly. OPT has the best performance since it optimally solves the optimization problem. However, GRD also has a nice performance, even though it does not always guarantee the optimal. DP-1 performs slightly better than DP-3 since DP-1 knows the global data popularity while DP-3 is based on dynamic popularity within a small window. RAND performs worst, especially when the number of requests is large. With the increase in the number of requests, we anticipate the difference between DP-1 and DP-3 will gradually shrink, and the difference with random strategy will become larger.

Then, we consider the scenario defined in Section IV-C, where data placement is performed periodically. In this case, we measure not only the accessing cost but also the placement cost during the placement phase (as defined in Equ. (5)). Fig. 5(b) shows the average total cost of OPT and GRD compared with RAND. The results are similar to those in Fig. 5(a), where OPT and GRD perform better than RAND. Also, since including the placement cost, the average total cost of the same schemes is larger than that of the average accessing cost. It is clear that our proposed OPT method can handle the new optimization problem well.

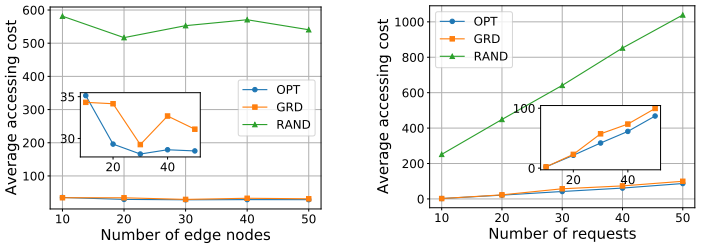
Next, we evaluate the performance of the proposed solutions via simulations based on trace-driven mobility data [15]. The number of edge servers is first set to 30 and the user mobility follows the trace-driven mobility data. We perform the simulation under the 30-minute period. Fig. 6(a) displays the accessing costs of different methods in different time slots. Clearly, OPT and GRD perform the best and have similar near-optimal results, while RAND performs worst with much higher accessing costs. As shown in the zoom-in subplot, OPT achieves less accessing costs than GRD in most cases. For the optimization with the total cost where both accessing and deployment costs are considered, Fig. 6(b) shows similar results, i.e., both OPT and GRD still perform much better than RAND does and OPT outperforms GRD at certain time slots.

Last, we also investigate the effect of different numbers of



(a) accessing cost (b) total cost

Fig. 6. **Optimization based methods:** Comparison of proposed solutions under different time slots, when (a) only accessing cost is considered, or (b) both accessing cost and placement cost are considered.



(a) different # of edge servers (b) different # of data requests

Fig. 7. **Optimization based methods:** Comparison of proposed solutions under different edge servers and data requests.

edge servers and data requests using the trace-driven simulation. The results reported in Fig. 7(a) are from the simulations where the number of edge servers N ranges from 10 to 50 and the maximal number of data requests U is set to 30. In terms of edge servers connection, the link between edge servers are generated randomly. Here we consider data placement for a single time slot and only the accessing cost. From Fig. 7(a), we can see that the average cost does not change with the increasing of the number of edge servers. The advances of OPT and GRD are consistent. Fig. 7(b) shows the results of simulations under different number of data requests U. It is obvious that with more data requests, more overall accessing cost is needed. The overall trends are similar to those in Fig. 5.

VI. Conclusion

In this paper, we study the data placement problem in the edge clouds while data items have various popularity. We mainly propose two types of data placement schemes, i.e., data popularity-based and optimization-based data placement. When the data requests are unknown, our data popularity-based scheme maps data items and edge servers to a virtual plane based on data popularity and network delay, respectively, and then the data placement is based on the virtual coordinates. When the data requests are known, an optimization-based scheme can be used to minimize the total accessing cost (and placing cost). Our simulations confirm the efficiency of the proposed algorithms compared with the existing methods.

It is obviously when the data requests are known, optimization-based solution gives better solution as shown in Fig. 5. However, in practice, it is difficult to predict future data requests, therefore, virtual-space-based method can be used due to its simplicity. Our proposed data popularity-based data placement schemes outperform the existing virtual-space methods since it takes data popularity into the consideration. In addition, it is clear that these two types of methods (optimization-based vs data popularity) are complementary. The appropriate method should be selected for different usage scenarios. In the future, we plan to further investigate how to combine the proposed data placement methods with efficient computing task scheduling (such as [7], [8], [16]–[18]), routing [19], and privacy protection (such as [20], [21]), to better serve data-intensive edge computing.

References

[1] W. Shi, J. Cao, et al., "Edge computing: Vision and challenges," *IEEE Internet of Things J.*, vol. 3, no. 5, pp. 637–646, 2016.

[2] C. Li, J. Tang, et al., "Collaborative cache allocation and task scheduling for data-intensive applications in edge computing environment," *Future Generation Computer Systems*, vol. 95, pp. 249–264, 2019.

[3] V. Farhadi, et al., "Service placement and request scheduling for data-intensive applications in edge clouds," in *IEEE INFOCOM*, 2019.

[4] C. Hamdeni, T. Hamrouni, and F. B. Charrada, "Data popularity measurements in distributed systems: Survey and design directions," *Journal of Network and Computer Applications*, vol. 72, pp. 150–161, 2016.

[5] Y. Shao, C. Li, and H. Tang, "A data replica placement strategy for iot workflows in collaborative edge and cloud environments," *Computer Networks*, vol. 148, pp. 46–59, 2019.

[6] B. Lin, et al., "A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing," *IEEE Trans. on Industrial Informatics*, vol. 15, no. 7, pp. 4254–4265, 2019.

[7] C. Li, J. Bai, and J. Tang, "Joint optimization of data placement and scheduling for improving user experience in edge computing," *Journal of Parallel and Distributed Computing*, vol. 125, pp. 93–105, 2019.

[8] M. Breitbach, D. Schäfer, et al., "Context-aware data and task placement in edge computing environments," in *IEEE PerCom*, 2019.

[9] J. Xie, C. Qian, et al., "Efficient indexing mechanism for unstructured data sharing systems in edge computing," in *IEEE INFOCOM*, 2019.

[10] J. Xie, C. Qian, et al., "Efficient data placement and retrieval services in edge computing," in *IEEE ICDCS*, 2019.

[11] Y. Wang and X.-Y. Li, "Efficient Delaunay-based localized routing for wireless sensor networks," *Wiley International Journal of Communication System*, vol. 20, no. 7, pp. 767–789, 2007.

[12] Y. Wang, C.-W. Yi, M. Huang, and F. Li, "Three dimensional greedy routing in large-scale random wireless sensor networks," *Ad Hoc Networks Journal*, vol. 11, no. 4, pp. 1331–1344, 2013.

[13] S. S. Lam and C. Qian, "Geographic routing in d-dimensional spaces with guaranteed delivery and low stretch," *IEEE/ACM Transactions on Networking*, vol. 21, no. 2, pp. 663–677, 2013.

[14] PuLP 2.3. [Online]. Available: https://pypi.org/project/PuLP/

[15] Mongnam Han, Youngseok Lee, Sue B. Moon, Keon Jang, Dooyoung Lee, CRAWDAD dataset kaist/wibro (v. 2008-06-04). [Online]. Available: https://crawdad.org/kaist/wibro/20080604

[16] X. Wei and Y. Wang, "Joint resource placement and task dispatching in mobile edge computing across timescales," in *IEEE/ACM IWQoS*, 2021.

[17] S. Yang, N. He, F. Li, et al., "Survivable task allocation in cloud radio access networks with mobile edge computing," *IEEE Internet of Things J.*, vol. 8, no. 2, pp. 1095–1108, 2021.

[18] S. Yang, F. Li, M. Shen, et al., "Cloudlet placement and task allocation in mobile edge computing," *IEEE Internet of Things J.*, vol. 6, no. 3, pp. 5853–5863, 2019.

[19] S. Yang, F. Li, S. Trajanovski, et al., "Delay-aware virtual network function placement and routing in edge clouds," *IEEE Trans. on Mobile Computing*, vol. 20, no. 2, pp. 445 – 459, 2021.

[20] T. Li, Z. Qiu, et al., "Privacy-preserving participant grouping for mobile social sensing over edge clouds," *IEEE Trans. on Network Science and Engineering*, vol. 8, no. 2, pp. 865 – 880, 2021.

[21] Y. Liu, T. Feng, et al., "DREAM: Online control mechanisms for data aggregation error minimization in privacy-preserving crowdsensing," *IEEE Trans. on Dependable and Secure Computing*, to appear.