Popularity-Based Data Placement With Load Balancing in Edge Computing

Xinliang Wei, *Student Member, IEEE* and Yu Wang, *Fellow, IEEE*

Abstract—In recent years, edge computing has become an increasingly popular computing paradigm to enable real-time data processing and mobile intelligence. Edge computing allows computing at the edge of the network, where data is generated and distributed at the nearby edge servers to reduce the data access latency and improve data processing efficiency. One of the key challenges in data-intensive edge computing is how to place the data at the edge clouds effectively such that the access latency to the data is minimized. In this paper, we study such a data placement problem in edge computing while different data items have diverse popularity. We propose a popularity based placement method which maps both data items and edge servers to a virtual plane and places or retrieves data based on its virtual coordinate in the plane. We then further propose additional placement strategies to handle load balancing among edge servers via either offloading or data duplication. Simulation results show that our proposed strategies efficiently reduce the average path length of data access and the load-balancing strategies indeed provide an effective relief of storage pressures at certain overloaded servers.

Index Terms—Data placement, data popularity, load balancing, data replication, edge computing

1 Introduction

With the increasing amount of data generated by diverse applications and devices, especially the large amount of data collected by Internet of Things (IoT) services or generated by smart devices, data transmission has become the bottleneck of traditional cloud computing platforms. Sending all the data to the cloud platform 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 pressure. In addition, with the advance of Artificial Intelligence of Things (AIoT), not only millions of data are generated from daily smart devices, such as smart light bulbs, smart cameras, various sensors, but also a large number of parameters of complex machine learning models have to be trained and exchanged by these AIoT devices. Classical cloud-based platforms have difficulty communicating and processing these data/models effectively with sufficient privacy and secure protection. This has further accelerated the growth of this new computing paradigm - edge computing [1].

As shown in Fig. 1, a typical edge computing environment consists of several entities: mobile user, edge server, edge network and remote cloud. Unlike the cloud environment, edge servers are geographically dispersed at the edge of the network near the mobile users and own heterogeneous computing and storage capability [1], [2], [3], [4], [5], [6], [7]. Each edge server can provide services for those mobile users in the specific nearby area by holding some data/models and performing the computation task based on data/models. Hereafter, we use data to refer to both data and models as long as they are required for performing the service requested by mobile users.[1] When a mobile user requests data, its request is forwarded to the nearest edge server. If the edge server has the data, it can response the mobile user immediately with the data (as Data C in Fig. 1) or perform the corresponding computing service for the user. Otherwise, the edge server has to retrieve the data from other edge servers (Data A or B) or even from the remote cloud (Data F). Clearly, the 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 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 B) 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.

Data placement has been well studied in distributed systems [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26]. However, edge computing has its own characteristics [1], such as proximity, fluctuation, and heterogeneity. Edge servers deployed in the edge network are in the proximity of mobile users compared with the distributed system (e.g., cloud computing). So it improves the speed of data processing as a direct result of lower latency.

• *The authors are with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19112 USA. E-mail: {xinliang.wei, wangyu}@temple.edu.*

Manuscript received 10 Aug. 2020; revised 11 May 2021; accepted 5 July 2021. Date of publication 13 July 2021; date of current version 8 Mar. 2023. (Corresponding author: Yu Wang.) Recommended for acceptance by F. Ye. Digital Object Identifier no. 10.1109/TCC.2021.3096467

1. Here, we do not differentiate the personal data or public data, as long as the data/model will be used/shared by multiple users at different locations. Also different security and privacy protection techniques [8], [9], [10] can be applied before the data placement.

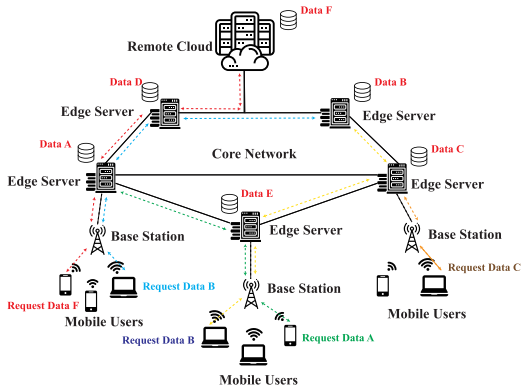


Fig. 1. A typical edge computing environment where data (or models) are placed at edge servers and shared by mobile users.

In addition, devices are usually user-controlled and can leave the edge network at any time. That means the network status is fluctuating over time. Furthermore, the topology of edge environments is heterogeneous and dynamic, which will bring another challenge to the data placement, e.g., how to maintain the existing data already stored in the edge server when the topology is changed. Thus data placement problem in edge computing has also drawn significant attentions from researchers recently [27], [28], [29], [30]. But most of them formulate the data placement problem as an optimization problem and leverage complex optimization solvers to tackle it. Such methods suffer from high computation and communication overheads, which makes them not suitable for large-scale systems. Most recently, Xie *et al.* [31], [32] proposed a novel virtual space based method, which maps both switches and data indexes/items into a virtual space and places data based on virtual distance in the space. Their method can enable efficient retrieve via greedy forwarding. However, none of them consider data popularity when placing data on edge servers in edge computing. Therefore, this work aims to verify the feasibility of considering data popularity in data placement problems.

In this paper, we investigate the static data placement strategy based on data popularity in edge computing with the aim of reducing the average forwarding path length of data. Inspired by [31], [32], we also adopt a virtual-space based placement method with greedy routing-based retrieve, 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 and balance the load among edge servers, we further propose several placement strategies which either offload data items to other servers when the assigned server is overloaded or place multiple replicas of the same data item to reduce the assigned load of servers. In both cases, we do take data popularity into consideration when designing the offloading and replication strategies. Simulation results show that our proposed strategies can achieve better performance compared to existing solutions [31], [32]. Moreover, both the offloading and replication strategies can effectively handle the storage pressure of overloaded edge servers.

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

- To our best knowledge, our proposed 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 retrieve process and reduces the overall response latency.
- We also propose several offloading and replication strategies which can make smart offloading and replication decisions based on data popularity, to further reduce the pressure on overloaded servers and improve the overall performance.
- We have conducted extensive simulations to verify the efficiency and effectiveness of the proposed data placement strategies. It confirms the advance of taking data popularity into consideration in our design.

Even though our proposed solution is mainly designed for the static data placement, it also enables potentially quick adaptation to handle dynamic edge environment. For example, a new data item or an update of data popularity of an existing data item can be easily handled with a new calculation of its coordinates in the virtual plane. If there is an update of network topology, the edge servers' coordinates need to be regenerated, which can be efficiently handled by the controller of software-defined edge network infrastructure. Our virtual-space based solution is much more agile than the optimization-based data placement methods.

The rest of this paper is organized as follows. Section 2 provides related works on data placement. Section 3 introduces the motivation of our study and the overall design of our method. Section 4 presents the virtual coordinate construction of the edge server and data which plays the center role in our design. Section 5 describes the detail of our proposed data placement and retrieval methods. Sections 6 and 7 present additional data placement strategies with the consideration of storage limits and multiple replicas, respectively, to balance the load. Evaluations of proposed methods are provided in Section 8. Finally, Section 9 concludes the paper with possible future directions.

2 Related Works

Data placement has been an important topic in distributed database/system [11], [12], [13], peer-to-peer networking [14], [15], [16], [17], content delivery network[18], [19], and cloud computing[20], [21], [22], [23], [24], [25], [26]. Due to the similarity of cloud computing and edge computing, here, we mainly focus on reviewing recent data placement strategies in cloud computing. Li *et al.* [20] proposed a clustering algorithm based on the principle of minimum distance to dynamically place data into clusters and a consistent hashing method to decide the specific storage servers to hold the data in cloud computing. However, the proposed methods only consider the similarity among data as the distance metric in clustering but ignore all networking and computing delay

among servers. Xu *et al.* [21] investigated the data placement problem among data centers via cloud computing, and proposed a genetic algorithm to obtain the best approximation of data placement. Similarly, Guo *et al.* [22] also used a genetic algorithm to solve the data placement among data centers where both the cooperation costs among data slices and the global load balancing are considered. Wang *et al.* [23] also studied data placement strategy for data-intensive computations in distributed cloud systems, which aims to minimize the total data scheduling between data centers while maintaining statistical I/O load balancing and capacity load balancing. In addition, several works [24], [25], [26] have focused on the data placement strategy for scientific workflows in cloud environment where data dependency and temporal relation among data and tasks play important roles. However, these centralized data placement methods suffer from high computation and communication overheads.

While similar to cloud computing, edge computing has its own characteristics [1]. Thus the data placement problem in edge computing becomes a new emerging topic in recent years. Both Shao *et al.* [27] and Lin *et al.* [28] have studied the data placement strategy for workflows in edge computing. Shao *et al.* [27] proposed a data replica placement strategy for processing the data-intensive IoT workflows which aims to minimize the data access costs while meeting the workflow's deadline constraint. The problem is modeled as a 0–1 integer programming problem to consider the data dependency, data reliability and user cooperation, and then solved by an intelligent swarm optimization. Similarly, Lin *et al.* [28] also proposed a self-adaptive discrete particle swarm optimization algorithm to optimize the data transmission time when placing data for a scientific workflow. Li *et al.* [29] investigated a joint optimization of data placement and task scheduling in edge computing to reduce the computation delay and response time. For the data placement optimization, the authors considered the value, transmission cost, and replacement cost of data blocks, and the formulated optimization problem is solved by a tabu search algorithm designed for the knapsack problem. However, again these optimization-based methods usually suffer from poor stability and high overheads. Breitbach *et al.* [30] have also studied both data placement and task placement in edge computing 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 is tuned based on context information (such as data size, remaining storage, stability, application).

Most recently, Huang *et al.* [33] have studied caching fairness for data sharing in edge computing environments. They propose fairness metrics to take resources and wireless contention into consideration and formulate the caching fairness problem as an integer linear programming problem. Then they propose an approximation algorithm based on connected facility location algorithm and a distributed algorithm. Xie *et al.* [31] studied the data-sharing problem in edge computing and proposed a coordinate-based data indexing mechanism to enable the efficient data sharing in edge computing. It maps both switches and data indexes into a virtual space with associated coordinates, and then the index servers are selected for each data based on the virtual coordinates. Their simulations showed that both the routing path lengths and forwarding table sizes for publishing/querying the data indexes are efficient. Xie *et al.* [32] further extended their virtual-space method to handle data placement and retrieval in edge computing with an enhancement based on centrodial Voronoi tesselation to handle load balance among edge servers. Both [31] and [32] inspire our work on data placement with data popularity (adopting a virtual-space based placement method with greedy routing-based retrieve), but they do not consider the data popularity of data items.

Note that there are other types of resource management problems in edge computing, such as virtual network function placement [34], [35], service placement [36], [37], and cloudlet placement [38], [39], [40]. These problems are different from the data placement problem, and their solutions could not solve the considered data placement problem here.

3 Data Popularity and Design Overview

3.1 Network Models and Data Placement Problem

In this paper, we consider a typical edge computing environment as shown in Fig. 1, where an edge network $G(V, E)$ connects N edge servers with M links. Here $V = \{v_1, v_2, \ldots, v_N\}$ and $E = \{e_1, e_2, \ldots, e_M\}$ denote the set of edge servers and the set of direct links among them, respectively. For each edge server v_i, we assume that it has a specific maximal storage capacity $c_i = c(v_i)$. Let $l_{ij} = l(v_i, v_j)$ to represent the shortest path length from edge server v_i to v_j in G, we then have a distance matrix $L = \{l_{ij}\}$ which holds lengths of all shortest paths in the edge network. Assume that we have W data items, $D = \{d_1, d_2, \ldots, d_W\}$, in the system. Each data d_i has a specific data size $s_i = s(d_i)$ and data popularity $p_i = p(d_i)$ (which will be explained in the next subsection). For each of data item d_i, we need to find an edge server v_j to hold it. Then the data placement 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) to stored data items in edge network G and also balance the load among edge servers. Xie *et al.* [32] proposed a nice virtual-space based data placement strategy for edge computing problem however they did not consider data popularity among data items d_i. Compared with complex optimization-based data placement strategy, the virtual space based method is much simple and easy to implemented.

3.2 Data Popularity

Data popularity measures how much a given piece of data is requested by the users in a system. This gives an indication of the importance of that data. Therefore, it is one of the most important parameters in the design of various data-centric distributed systems and enable more intelligent data management, such as file assignment in parallel I/O system [41], replication management in distributed storage systems [42], [43], load balancing in content delivery networks [44], and coordinated caching in named data networking [45]. Hamdeni *et al.* [46] provided a nice survey on data popularity and highlighted its importance in the replication management in distributed systems. It is clear that taking popularity into account allows to better place the data or

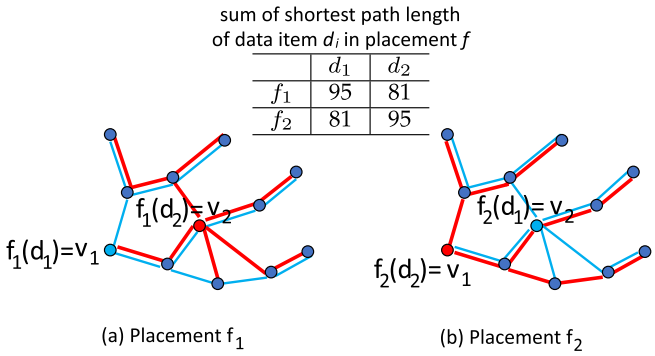


Fig. 2. Examples of two placement strategies of two data at two servers.

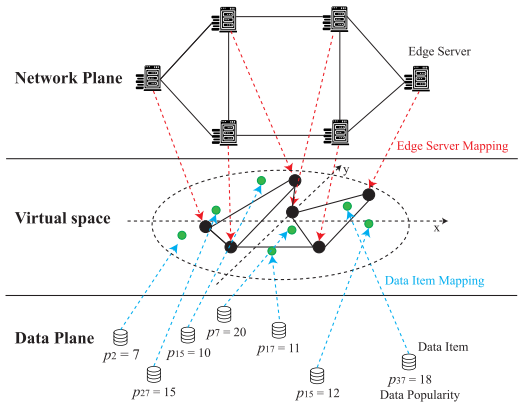


Fig. 3. Virtual space based approach: edge servers and data items are mapped to a virtual space and associated with virtual coordinates.

their replicas to avoid overloaded sites in any distributed systems.

For the data placement problem in edge computing, data popularity is also critical. First, data placement aims not only to minimize the data access delay but also load balancing among edge servers, thus data items have to be place among different servers. 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. For example, Fig. 2 shows an example of two placement strategies, which place two data items d_1 and d_2 at servers v_1 and v_2 respectively and differently. Assume that $p(d_2) > p(d_1)$, thus there are more requests from other servers to d_2's location than d_1's. With different placement, the routes of shortest paths are different (as blue and red trees marked in the figure), and v_2 has shorter paths to all other servers than v_1 does. Table in the figure shows the total length of all shortest paths to each data item under two placement strategies. It is obvious that placement f_1 has better performance since the red shortest path tree has less path length (81) than the blue one (95) in Fig. 2a while reversed in Fig. 2b. Therefore, in this paper, 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 existing data placement strategies for edge computing do not consider data popularity. The only exception is [29], where the authors considered data popularity as part of their estimation of value of data block in their formulated placement problem. Particularly, they compute the data popularity based on the access frequencies of the data blocks and the time interval between two accesses, and use it as one of the parameters in their utility function. The data 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 data popularity, and then the placement decision is made based on the coordinates in the virtual space.

Data popularity can be assessed differently for distributed system depending on the application. In general, there are three factors contributing to the data popularity: the number of accesses (i.e., how many times the data item is requested), the life time, and the request distribution over time or space. In this paper we simply use the number of accesses as the data popularity. However, it is not difficult to extend our definition to include other two factors (or even other data popularity measurements) into our system. For each data item d_i, we assume that its data popularity $p_i = p(d_i)$ describes its number of access requests over time. We assume that data popularity for each data is known to the system. Larger data popularity means the data item is more frequently accessed by mobile users in the system. Obviously, the locations of the popular data items are at the roots of the overall data placement problem, compared with those of unpopular data. Note that there could also be more complex data popularity models, where various user or location specific preferences may be considered differently even for the same data item. Our proposed method may be further extended to deal with such models by treating the data preferences from different users/locations with different weights or more refined models. We leave such study as one of the future works.

3.3 Design Overview

Similar to [31], [32], our popularity-based data placement strategy adopts a virtual-space approach, which maintains a virtual space (i.e., a virtual 2D plane) and maps all edge servers and data items to such plane, as shown in Fig. 3. The data placement is based on the associated coordinates of edge servers and data items in the virtual plane. How to perform the mappings is critical in our design. When mapping edge servers from network plane into the virtual plane, we try to make sure the euclidean distance between two servers in the virtual plane is proportional to their real network distance. When mapping the data items from data plane into the virtual plane, we try to spread them out while taking into consideration of their data popularity such that the more popular data is closer to the center of the virtual space. The intuition behind of this design is the shortest paths to all other servers are shorter at the center area. Section 4 presents the detailed design of our coordinate construction algorithms. Given the constructed coordinates in the virtual space, we can make our placement decision simply based on the virtual distance between a data item and an edge server. The simplest version is to place the data item to the nearest server in the virtual plane. Section 5 describes our proposed placement and retrieval strategies.

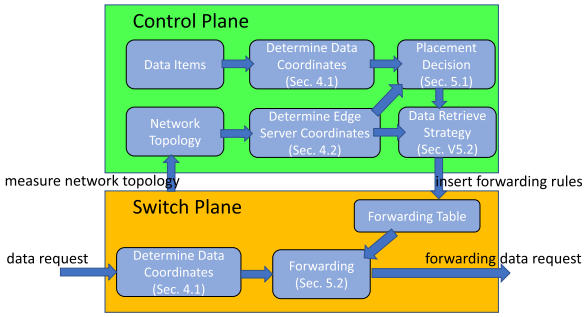
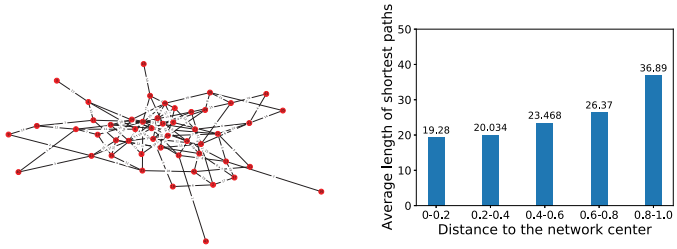


Fig. 4. Framework of the proposed data placement solution in a software-defined edge network infrastructure.



(a) network topology (b) shortest path length

Fig. 5. (a) An example of physical network topology of a random network with 50 edge servers. (b) The relation between the average length of shortest paths to other servers from a server and its distance to the network center.

Fig. 4 illustrates a framework of the proposed methods under a software-defined edge network infrastructure. In such a system, the switches provide data communication services to edge servers by following the forwarding rules/entries placed by the controller in the control plane. At the control plane, virtual coordinates are constructed for both data items and edge servers, and then a data placement strategy and its corresponding retrieval strategy will generate the forwarding rules to switches. At the switch plane, the switch first maps the data request to virtual plane and forward it based on its virtual coordinate and the installed forwarding rules.

To handle load balancing among edge servers and further reducing the data access delay, we propose several additional placement strategies (Section 6), which offload data items to nearby servers when the storage of desired edge server of our placement method exceeds its maximal limit, and new replica placement strategies (Section 7), which strategically place multiple replicas to serve users while considering the data popularity to decide the number of replicas with favor to more popular data.

In summary, in our design, data popularity places a central role. It has been considered during the construction of virtual coordinates of data items (Section 4), the selection of offloading choices (Section 6), and the decision on the number of replicas to deploy (Section 7).

4 Virtual Coordinate Construction

In this section, we discuss the construction of coordinates for both data and edge server in the virtual plane, which is a circular region with a radius of 1. The edge server and data will be mapped to this virtual space, and their coordinates will be unified to $[-1, 1]$. The center of the circular region (i.e., o with coordinates (0,0), as in Fig. 6) represents the center of the network. The virtual coordinate plane provides a viable alternative to geographic coordinates of edge servers and brings a new opportunity to link edge servers with data based on their virtual coordinates. Similar to [31], [32], mapping edge servers and data together to the virtual space enables much simpler data placement and retrieve (based on virtual coordinates, presented in Section 5).

4.1 Calculating Coordinates of Data Items Based on Data Popularity

Recall that each data d_i has data size $s(d_i)$ and data popularity $p(d_i)$. Assume that each data also has an unique identifier (index or ID) $ID(d_i)$. We compute the virtual coordinate of data by leveraging the Polar coordinate system and hash function. In the Polar coordinate system, each point is determined by a distance (r) and angle (θ) from a point and a direction respectively. In terms of hash function, given a specific key value with arbitrary size (in our case the data item's ID), a hash function can return a fixed-size hash value which we will use for generating the Polar coordinate of this data item in the virtual space.

Our proposed method to calculate the virtual coordinates of data has three goals. First, the mapping should be able to spread all data over the virtual plane where the edge servers will also sit. This can balance the load of data hosting among servers. Second, the mapping method needs to take data popularity into consideration and places the popular data items to the location which has smaller shortest paths to other regions. Last, the mapping method should be deterministic, i.e., given the same data item, the output of our mapping method should be the same. This can guarantee that for the data request on the same data item our retrieval process can lead to the same location in the virtual plane.

In our solution, we map each data d_i to a virtual location in the virtual plane whose polar coordinates are $r(d_i)$ and $\theta(d_i)$. To consider the data popularity in the mapping, our design is based on the following observation. In a dense network, the center area has smaller shortest paths to all regions. Fig. 5b shows that the average length of shortest paths to other servers and the distance to the network center for each server in a randomly deployed network with 50 servers (Fig. 5a). Clearly, the servers closer to the center of the network has less total length of all shortest paths. Based on this observation, our mapping method puts a popular data item near the center of the virtual plane. Specifically, we generate $r(d_i) \in [0, 1]$ using

$$r(d_i) = 1 - p(d_i)/p_{max}, \tag{1}$$

where p_{max} is the maximal data popularity among all data items. By doing so, the more popular data is, the closer to the center point as shown in Fig. 6a. To spread data items at different regions, we calculate the angle $\theta(d_i)$ using the hash value of the data's ID. Particularly, we first calculate the hash value $H(d_i)$ by using a hash function H (e.g., SHA-256). Next, we reduce the hash value to the scope of the virtual space by (1) using only the last 4 bytes of $H(d_i)$ and

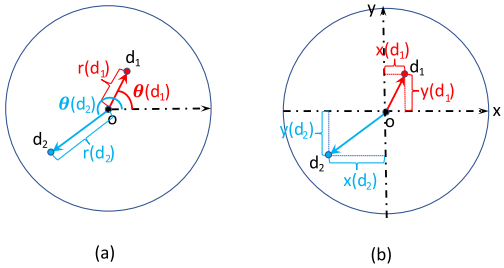


Fig. 6. Illustration of the virtual coordinates of two data items in the virtual plane with (a) Polar coordinates and (b) Cartesian coordinates. Here d_1 has a higher data popularity than d_2.

converting them to a 4-byte binary value $h(d_i)$, and (2) normalizing $h(d_i)$ between 0 and 2π. In other words,

$$\theta(d_i) = 2\pi \times h(d_i)/(2^{32} - 1). \tag{2}$$

By doing so, we place this data items along certain direction in Polar coordinates. Different data items will be spread all different directions. Even the data items with the same data popularity will be placed at different locations. The final polar coordinates are $(r(d_i), \theta(d_i))$, whose corresponding Cartesian coordinates can be obtained by

$$\begin{cases} x(d_i) = r(d_i) \times \cos\theta(d_i) \\ y(d_i) = r(d_i) \times \sin\theta(d_i). \end{cases} \tag{3}$$

Fig. 6 illustrates the relationship of virtual coordinates between Polar and Cartesian coordinates. All data items are mapped into a circular region with unit radius in the virtual plane. The construction of coordinates for all data items can be done in $O(W)$, where W is the number of data items.

4.2 Calculating Coordinates of Edge Servers Based on Network Distance

We also want to spread all edge servers in the same virtual plane. The major goal of the mapping of edge servers is to make sure that the euclidean distance between two edge servers in the virtual plane is proportional to their physical network distance. By doing so, when we place popular data items near the center in the virtual plane (as in Section 5.1), the accessing cost of them will be relevantly smaller since the cost is proportional to the distance in the virtual plane. In addition, this will ensure the local retrieve proposed in Section 5.2 (which picks the next server based on their virtual coordinates) has low routing stretch. This mapping problem is basically a network embedding (or graph embedding) problem, which has been well studied. Given the network topology G and the shortest path measurements among edge servers, we adopt the M-position algorithm used by [31], [32] to generate the virtual coordinates of edge servers in the 2D virtual plane. For the completeness, we briefly review the basic idea of such an algorithm. Given the network topology G, we can obtain the shortest path matrix $L = \{l_{ij}\}$, where l_{ij} is the shortest path length from edge server i to j. Using L as the input, the M-position algorithm aims to calculate the coordinates of edge servers, which can be represented as a coordinate matrix Q (a $2 \times N$ matrix of N edge servers in the two dimensional virtual plane), i.e.,

$$Q = \begin{bmatrix} x(v_1) & x(v_2) & \cdots & x(v_N) \\ y(v_1) & y(v_2) & \cdots & y(v_N) \end{bmatrix}.$$

The key idea behind the M-position algorithm is based on the fact that Q can be derived from a scalar product matrix $B = \frac{1}{2} JL^{(2)}J$ via the eigenvalue decomposition [31]. The major steps of the mapping algorithm to generate coordinates of edge servers is given as follows.

1) Given the network topology G, generate the shortest path matrix $L = \{l_{ij}\}$ and compute $L^{(2)} = \{l_{ij}^2\}$, which is the squared distance matrix.
2) Compute the scalar product matrix $B = \frac{1}{2} JL^{(2)}J$, where $J = 1 - \frac{1}{N}A$ and A is an $N \times N$ matrix of ones.
3) Determine two largest eigenvalues λ_1, λ_2 and the corresponding eigenvectors ξ_1, ξ_2 of matrix B.
4) Construct the coordinates of edge servers $Q = \Xi_2 \Lambda_2^{1/2}$, where Λ_2 is the matrix of two eigenvalues and Ξ_2 is the matrix of two eigenvectors of the matrix B.
5) Normalize Q to $\frac{1}{\sqrt{2} q_{max}} Q$, where q_{max} is the largest absolute value of all elements in Q, so that all coordinates of edge servers are within the circular region with unit radius in the virtual plane.

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.

5 Data Placement and Retrieve

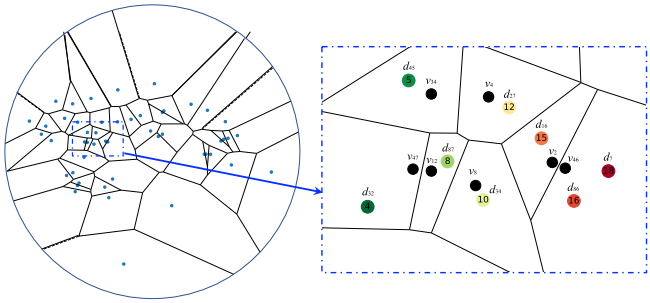
In this section, we discuss how our data placement strategy places and retrieves data based on the virtual coordinates.

5.1 Placing Data to Edge Servers

Since we have obtained the coordinates of both edge servers and data items in the same region on the virtual plane, the data placement becomes quite straightforward. Here, we first assume that each edge server has sufficient storage to hold the data placed by our placement strategy. We will discuss how to handle load balancing when there is a storage limit at edge server in Section 6. Our proposed basic data placement strategy (denoted by *Basic_Placement*) places each data item d_i to the edge server that has the nearest euclidean distance to this data item in the virtual plane, i.e.,

$$\begin{aligned} f(d_i) &= \arg\min_{v_k} ||v_k, d_i|| \\ &= \arg\min_{v_k} \sqrt{(x(v_k) - x(d_i))^2 + (y(v_k) - y(d_i))^2}. \end{aligned}$$

Fig. 7 show examples of data placement output. Clearly, the assignment of servers forms a Voronoi diagram (Fig. 7a) in the region, where if a data item falls within a Voronoi cell then it will be placed at the edge server who owns the Voronoi cell. Since the more popular data items are more towards the center of the network (as shown in Fig. 7b), they will be placed to edge servers whose shortest paths to other servers are shorter. It is noted that the center of the network is relative to the virtual plane, which is the center of the circular region. The popular data items are placed to the servers near to the center of the network, which are determined based on the distances on this virtual plane. The data placement decision is made within $O(WN)$ where N and W are the number of edge servers and data items, respectively.



(a) Voronoi diagram of all edge servers in virtual plane

(b) a zoom-in view of data placement in Voronoi cells

Fig. 7. Illustration of basic data placement: (a) Voronoi diagram formed by all edge servers; (b) a zoom-in view to see examples of locations of data items based on their popularity. Here, black dots are edge servers, while other dots are data items whose colors reflect their popularity (the more red the color, the more popular the data). The numbers inside dots are popularity values. More popular data is placed nearer to the center (on the left of the rectangle).

5.2 Retrieving Data from Edge Servers

With our proposed coordinate construction and data placement, there are two possible retrieve strategies to retrieve a data from the edge network: *global retrieve* and *local retrieve*.

Global Retrieve: Global retrieve method assumes that edge server and network controller have the global knowledge of the topology. Based on this knowledge, the shortest paths between any pair of servers are calculated and corresponding switching information will be deployed in underlying switches. When a mobile user at edge unit requests a data item d_i, it first determines the coordinate $(x(d_i), y(d_i))$ of this data in virtual plane based on its index and popularity. Then based on our placement strategy, it can calculate the target server which holds the data item, i.e., $v_j = f(d_i)$. Next, the data request is routed towards v_j by following the stored shortest path until it reaches the target server. Global retrieve strategy not only can guarantee the delivery of data request, but also has the minimal retrieval latency since it relays the request over the shortest path. However, the drawback is the additional storage overhead at switches and edge servers to store the shortest paths to all edge servers.

Local Retrieve: In contrast, local retrieve method assumes that each edge server only knows the information from its neighboring edge servers. Thus it enjoys lower storage at servers and switches. However, the challenge is how to route the request towards the target server. Note that instead of finding the target server $f(d_i)$ directly (which needs the knowledge of coordinates of all servers), in local retrieve, the data request is routed towards the coordinate $(x(d_i), y(d_i))$ of the data item in virtual plane. At each server when receiving the request, it first checks whether this data is placed in itself. If the current edge server is the target server, it will reply the request. Otherwise, the current server greedily selects the next server to forward from its neighboring server based on their coordinates. The criteria is to pick up the server whose coordinate is nearest to the target coordinate $(x(d_i), y(d_i))$ in the virtual plane. This procedure repeats until the request reaches the target server. Notice that greedy forwarding may fail at local minimum, but randomized forwarding can be used to get out of the local minimum[2] However, compared with the global retrieve, local retrieve causes longer retrieval latency which is due to (1) longer exploration process to find the target server and (2) longer founded delivery path between source and target servers.

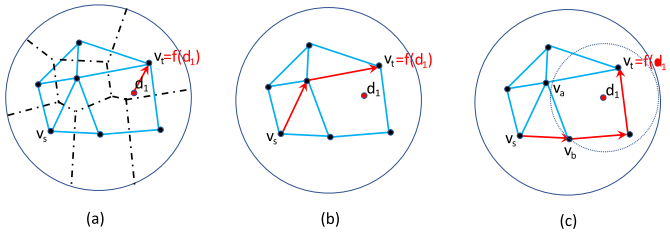


Fig. 8. Illustration of global retrieve versus local retrieve: (a) placement of d_1 to v_t based on the placement strategy, (b) global retrieve via the global shortest path to v_t, (c) local retrieve towards location of d_i at $(x(d_i), y(d_i))$. Note that v_b is closer to $(x(d_i), y(d_i))$ than v_a, thus local retrieve takes a different route than the global shortest path.

Fig. 8 illustrates the difference between these two retrieve methods. Clearly, there is a trade-off between the performance (retrieve latency) and the complexity (computing, storing and updating the global shortest paths). Our proposed data placement method supports both retrieve methods, and can select the appropriate one based on different application scenarios.

6 Data Placement With Limited Storage

We have introduced the basic data placement strategy based on data popularity in previous section. However, we did not consider the storage capacity at edge server yet. Based on the basic placement strategy some of the edge servers may be overloaded with data items. If each edge server v_i has specific maximal storage capacity $c_i = c(v_i)$ such that it can only stores data items whose total size is up to c_i. Hereafter, we use $cc(v_i)$ to denote the current storage usage of server v_i. In this section, we propose some simple heuristics to handle load balancing[3] due to storage limits. All the heuristics use the output (i.e., $f(d_i)$) of our basic data placement (Section 5.1) as their input, but they are different with each other in (1) the ordering of data placement and (2) the choice of offloading server. Here, this offloading is just an additional step during the making of data placement decision. The real placement of data items on servers happened after the final data placement decision. In addition, due to the offloading, data retrieve also needs to be able to find the offloading server.

6.1 Processing Order for Data Placement

After the basic data placement based on data popularity, we have a initial placement decision, which can be denoted as a list $place\_dec$ which consists of multiple two-tuple $(d_i, v_j = f_1(d_i))$, where d_i is the data index and v_j is the edge server assigned by the basic data placement for d_i. However, some of the edge server may not have enough storage to hold all assigned data items, therefore, we need make an offloading

2. There are other methods to eliminate the local minimum for greedy routing via adjusting transmission range [47] or building a Delaunay graph [48], [49].

3. Later in Section 7 we also consider using multiple replicas to further balance loads among servers. In addition, there are other possible load balancing strategies for edge computing [50], [51], [52], [53].

decision to assign some of their data items. However, which item from which server to offload is tricky since moving an assignment of a data item to a server may cause a chain effort on other servers. Such effort is also decided by the processing order of data placement with offloading. Here, we propose two different methods to process the data placement with offloading consideration.

Algorithm 1. Data Placement in Order of Data Popularity

Input: The placement decision $place\_dec = \{(d_i, f_1(d_i))\}$ determined by **Basic_Placement**.
Output: The updated new placement decision $place\_dec$.
Sort $place\_dec$ in descending order of popularity $p(d_i)$;
for each $item = (d_i, v_j)$ in $place\_dec$ **do**
  if $s(d_i) + cc(v_j) > c(v_j)$ **then**
    $v_l =$ **Find_Offloading_Server**(d_i, v_j);
    Update and confirm $item = (d_i, v_l)$, i.e.,$f_2(d_i) = v_l$;
    $cc(v_l)$ += $s(d_i)$;
  else
    Confirm $item = (d_i, v_j)$, i.e.,$f_2(d_i) = v_j$;
    $cc(v_j)$ += $s(d_i)$;
return $place\_dec = \{(d_i, f_2(d_i))\}$

Algorithm 2. Data Placement in Order of Server Capacity

Input: The placement decision $place\_dec = \{(d_i, f_1(d_i))\}$ determined by **Basic_Placement**.
Output: The updated new placement decision $place\_dec$.
Sort V in descending order of server capacity $c(v_i)$;
for each v_i in V **do**
  Generate D_i based on $place\_dec$;
  Sort D_i in descending order of data popularity $p(d_k)$;
  for each d_k in D_i **do**
    if $s(d_k) + cc(v_i) > c(v_i)$ **then**
      $v_l =$ **Find_Offloading_Server**(d_k, v_i);
      Update/confirm d_k's placement, i.e.,$f_3(d_k) = v_l$;
      $cc(v_l)$ += $s(d_k)$;
    else
      Confirm d_k's placement, i.e.,$f_3(d_k) = v_i$;
      $cc(v_i)$ += $s(d_k)$;
return $place\_dec = \{(d_k, f_3(d_k))\}$

Placing Data in the Order of Popularity The first method processes the data item based on their popularity to confirm the data item placement or offload it to other server. It first sorts the resulting list $place\_dec$ in descending order based on data popularity $p(d_i)$. Next, for each item $(d_i, v_j = f_1(d_i))$ in $place\_dec$, if the summation of data size $s(d_i)$ of d_i and the current server storage $cc(v_j)$ of v_j does not exceed the maximal server storage $c(v_j)$, then we confirm this placement and place this data d_i to this edge server v_j. Otherwise, we find an available edge server v_l to offload this data item (denoted by a procedure *Find_Offloading_Server*) and modify the initial placement decision $f_2(d_i) = v_l$. Multiple ways to find such edge server to offload will be discussed in the next subsection. The detailed algorithm is presented in Algorithm 1. By performing this algorithm, we can guarantee that each server has sufficient storage to hold all assigned data items and avoid the overloading of certain edge servers. The total time complexity of Algorithm 1 is $O(W\log W + WX)$, where $O(W\log W)$ is from ordering the data popularity and $O(WX)$ is for W rounds of finding offloading server for each data item. Here, X is the cost of *Find_Offloading_Server*, which is bounded by the number of neighbors of the server v_i or by N depending which method is used.

Placing Data in the Order of Server Capacity The second method processes the data placement in a different order which is based on the maximal edge server storage capacity. The idea of this strategy is to deal with the edge server that has bigger maximal storage capacity first. When determining which data should be placed on the current edge server and which should be offloaded, we continue to take into account data popularity, where more popular data is easier to stay at the current server. The algorithm acts as follows. First, we sort the list of edge servers V in descending order according to the maximal edge server storage capacity $c(v_i)$, such that $c(v_1) \geq c(v_2), \ldots, \geq c(v_N)$. For each server v_i, we define D_i as a list which consists of all data items assigned to v_i by the basic data placement, i.e., $D_i = \{d_k | v_i = f_1(d_k)\}$. Then we process the edge server in order to confirm or update the data placement on that server. For each server v_i, D_i is sorted based on data popularity in descending order. We process the data item $d_k \in D_i$. If placing this item does not exceed the maximal storage of v_i, i.e., $s(d_k) + cc(v_i) \leq c(v_i)$, its placement is confirmed. Otherwise, we simply call the procedure *Find_Offloading_Server* to find a near server to place it and update its placement $f_3(d_k)$. The whole process is repeated for all data items on all servers. The detailed algorithm is presented in Algorithm 2. The major difference with the first method is that the processing order is based on server capacity (the outer "for" loop in Algorithm 2). The total time complexity of Algorithm 2 is $O(N\log N + \sum_{i=1}^{N}(|D_i|\log|D_i| + |D_i|X))$. Here, $O(N\log N)$ is from ordering the server capacity, $O(|D_i|\log|D_i|)$ is from ordering the data popularity in D_i, and $O(|D_i|X)$ is $|D_i|$ rounds of find offloading server for each data item in D_i. Note that $\sum_{i=1}^{N}|D_i| = W$.

6.2 Offloading Choice

Now we discuss the two possible methods to implement the procedure *Find_Offloading_Server*(d_i, v_j) to find the offloading server v_l for data item d_i at v_j (since v_j does not have sufficient storage).

The first method (*Nearest_Neighbor*) simply finds the offloading server from v_j's neighboring edge servers in topology G. The selection criteria are (1) $v_l \in \{\overline{v_i v_l} \in E\}$; (2) $s(d_i) + cc(v_l) \leq c(v_l)$; and (3) $||v_l, v_i||$ in the virtual plane is the minimum among all candidates satisfying (1) and (2). Note that if none of neighbors has sufficient storage, we search for neighbors' neighbors instead. This repeats until we can find a server to host d_i. *Nearest_Neighbor* aims to find a nearby server from v_j in the network topology to hold the data.

The second method (*Nearest_Server*) relaxes the requirement of v_l to be a neighbor of v_j, instead considering all possible servers in the region. It finds the offloading server based on (1) $s(d_i) + cc(v_l) \leq c(v_l)$ and (2) $||v_l, d_i||$ in the virtual plane is the minimum among all candidates satisfying (1). I.e., it picks the server which has enough storage and a minimal distance to d_i instead. *Nearest_Server* aims to find a server near to the data item d_i in virtual plane to hold it.

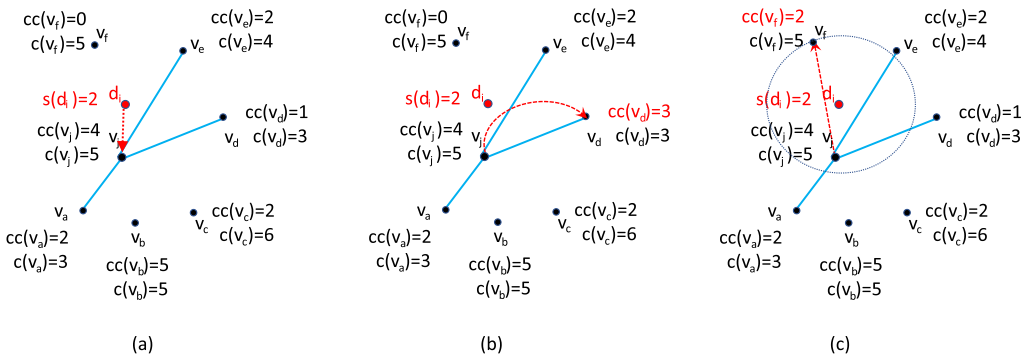


Fig. 9. Illustration of two methods for *Find_Offloading_Server*. (a) placement of d_i to v_j based on the placement strategy but v_j does not have sufficient storage; (b) *Nearest_Neighbor* where only neighboring servers (v_a, v_d, v_e) are considered, and v_d is selected since it is the nearest one to v_j with sufficient storage; (c) *Nearest_Server* where all surrounding servers are considered, and v_f is selected since it is the nearest one to d_i with sufficient storage.

Both methods try to offload d_i to a server nearby. The time X used to choose offloading server is bounded by the number of neighbors of the server v_i in G for *Nearest_Neighbor* or by N for *Nearest_Server*. Fig. 9 illustrates the difference between these two methods.

6.3 Data Retrieve

Since the proposed methods might offload data items from the original assigned server by $v_j = f_1(d_i)$ to another server, saying $v_l = f_2(d_i)$ or $f_3(d_i)$, we need have a way to let data retrieve method to find the new server. Note that even though the network controller may know the location of the new server, the each individual server when receiving the data item may not know the global information of server capability, thus fail to find the offloading decision. Instead broadcasting all offloading decisions to the whole network, a simple solution is to let the original server v_j host a forwarding entry to record the path towards the new server v_l. By doing so, the data retrieve methods can stay the same and the data request of d_i is still forwarded towards v_j. When it reaches v_j, it is then further forwarded to v_l. This will cost additional path length and retrieval latency. However, since the offloading server is selected to minimize the distance between v_j and v_l, the addition cost is minimized.

7 Data Placement With Multiple Replicas

While our proposed offloading methods can balance the data storage loads based on the maximal server storage, it does not solve the problem of overloading data requests to the servers. To address this problem, in this section, we propose new data placement strategy to balancing data request loads by leveraging replication. Data replication is the management of multiple replicas of the same data in the system. Replication strategies have been widely used in distributed systems [14], [17], [19], [27], [30], [42], [43] to ensure load balancing, reliability and data transfer speed as well as to offer the possibility to access the data efficiently from multiple locations.

In our data placement problem, if we allow multiple replicas of the same data distributed on different edge servers, it can not only provide load balancing but also improve the response time of data request and the delay of data delivery. However, how to design the replication strategy becomes crucial. Generally, different data characteristics and system conditions influence the replication strategy. The key challenge of implementing an effective replication strategy consists of two metrics. First, how to determine the number of replicas of each data items? Second, how to choose the edge server to place these replicas? Next we answer these two questions separately for our data placement design.

7.1 Number of Replicas

Inspired by [30], to efficiently determine the number of replicas, we take data size, data popularity, and the remaining storage capacity of all edge servers into account. The general ideas are (1) more replicas will be given to larger and more popular data items; and (2) more network storage available also lead to more replicas. For each of these aspects (i.e., data size, data popularity, and available storage), we normalize it with its maximal value in the network. Therefore, the number of replicas for each data d_i is calculated as follows.

$$n(d_i) = \left(\alpha_1 \frac{s(d_i)}{s_{max}} + \alpha_2 \frac{p(d_i)}{p_{max}} + \alpha_3 \frac{\sum_j (c(v_j) - cc(v_j))}{\sum_j c(v_j)}\right) \times \beta \times N. \tag{4}$$

Recall that N is the number of edge server, s_{max} and p_{max} are the maximal data size and data popularity. $cc(v_j)$ and $c(v_j)$ are the current used storage and maximal storage limit of v_j, thus $c(v_j) - cc(v_j)$ is the remaining storage capacity of v_j. α_1, α_2 and α_3 are weights added to these three coefficients, since the relative importance of three aspects can vary based on the data characteristics and system conditions. While $\alpha_1 + \alpha_2 + \alpha_3 = 1$, they can be adjusted to meet different requirements. For instance, with increasing data popularity, we can use higher α_2 to increase the number of replicas for the more popular data. Similarly, if our edge network system has limited storage capacity or low bandwidth, we may decrease α_3 or α_1 to meet the requirement. β is a ratio parameter to control the total number of replicas with respect to the total number of N. Larger β leads to more total replicas.

7.2 Placing Replicas

After we determine the number of replicas $n(d_i)$ of each data d_i, we have to determine where to place these replicas. The placement strategy of replicas should have the following goals. First, it should spread the replicas as evenly as possible

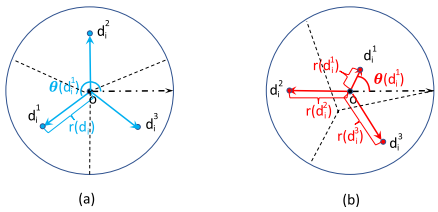


Fig. 10. Illustration of the virtual coordinates of replicas of data d_i generated by our method: (a) when $1-\frac{p(d_i)}{p_{max}} \geq \tau$ and (b) when $1-\frac{p(d_i)}{p_{max}} < \tau$. Here $n(d_i)=3$ and Voronoi cells of all replicas are shown in dashed lines.

in the region to balance the load. Second, the placement should still be popularity-aware, such that popular data has shorter shortest paths. Last, the mapping and placement methods should be non-randomized so that retrieve can be easily done. To achieve all of these, we modify our mapping method which generates the coordinates of data items (Section 4-A) to map a data item to $n(d_i)$ locations in the virtual plane based on its popularity. Both the placement and retrieve methods can still be the same, where each replica is just placed to the nearest server in the virtual plane and data requests are forwarding towards that server during retrieving.

Calculating Coordinates of Replicas. For each data item d_i, we will generate $n(d_i)$ data items, denoted by $d_i^1, d_i^1, \ldots, d_i^{n(d_i)}$, and spread them on the virtual plane. Inspired by the Voronoi diagram, we spread all replicas using the radius which depends on the data popularity but different angles. As shown in Fig. 10a, the Voronoi diagram formed by all replicas is evenly distributed in the virtual plane. The polar coordinates of kth replica d_i^k in the virtual plane are given by the following equations.

$$r(d_i^k) = 1-\frac{p(d_i)}{p_{max}}, \theta(d_i^k) = 2\pi\frac{h(d_i)}{2^{32}-1} + k\frac{2\pi}{n(d_i)}. \quad (5)$$

Note that the first copy of data item d_i^1 is mapped to the same location as d_i in Section 4.1. The radius and angle are still deterministically decided by data popularity and data index. The other replicas are evenly distributed by an angle difference of $\frac{2\pi}{n(d_i)}$ with the same radius. This solution seems achieve all desired goals, but it may have a problem when the data item is very popular. In that case, the radius is small, all replicas will be placed around the center of the network. Though their Voronoi cells are equal, this is not ideal since multiple replicas will be nearby. Therefore, we further modify the mapping method, by define a threshold $\tau < 0.5$. If $r(d_i^k) < \tau$, we shift $r(d_i^k)$ by adding 0.5, except for the first copy of the data which is still at original location. Fig. 10b shows such an example and the Voronoi cells of all replicas. Then, the new mapping method is given as follows.

$$r(d_i^k) = \begin{cases} 1.5-\frac{p(d_i)}{p_{max}}, & 1-\frac{p(d_i)}{p_{max}} < \tau \text{ and } k > 1 \\ 1-\frac{p(d_i)}{p_{max}}, & otherwise \end{cases}, \quad (6)$$
$$\theta(d_i^k) = 2\pi\frac{h(d_i)}{(2^{32}-1)} + k\frac{2\pi}{n(d_i)}.$$

In our simulations, we use $\tau = 0.01$. After we have the coordinates of all replicas, we can place these replicas to the closest edge server in virtual space. The retrieve procedure is straightforward too.

Our new placement method with replicas make sure that (1) the more popular data is, at least one copy of the data is closer to the center; (2) different data replicas are well spread in the virtual plane; (3) the shortest paths are reduced since copies of data can be find at multiple locations.

8 Performance Evaluation

In this section, we report the results from our simulations to evaluate our proposed data placement strategies.

8.1 Simulation Setup

To test our proposed data placement strategies, we randomly construct a network typology G with 50 edge servers whose degree satisfies a binomial distribution. The cost on each network link is set randomly from 1 to 20. Fig. 5a shows an example of such network topology. Since edge servers are not the normal servers with enough capacity, so we assume each edge server has a different limited maximal storage capacity ranges from 500MB to 1,000MB. In terms of data set, we randomly generate 100 data items with data size from 100MB to 150MB. To simulate the data popularity of each item, we leverage a real-world news popularity dataset [54] provided by University of Porto and randomly draw data popularity from it. Based on the simulated data popularity, we randomly generate data requests for different data items at random edge servers either in the whole region or at the boundary of network. Based on each data placement strategy, we place data items on corresponding edge server(s) and then perform data retrieves of all data items randomly from all edge servers based on their data popularity. Mainly, three performance metrics are used to evaluate the performance of proposed methods:

- *Average path length.* It is the average length of forwarding path of data requests from the source edge server to the target server during the data retrieve process. The length of a path is the summation of link costs of all links on the path, which reflects the end-to-end networking delay. Obviously, the shorter average path length the better data retrieve performance.
- *Average retrieval latency.* It is defined as the average running time of retrieve strategy during the data retrieval process. This mainly quantifies the complexity of the retrieval strategies. Here, latency is not due to the networking delay.
- *Distribution of number of data items.* To measure the load balancing of data placement, we also report the distribution of the number of data items on each edge server. The goal is to minimize the largest number of data items on single edge server.

We test seven different versions of our data placement strategies in the simulations, and they are

- *OUR-B.* This is the basic data placement strategy (*Basic_Placement*) proposed in Section 5. It does not consider the storage limit at each server and the placement decision is purely based on virtual coordinates.
- *OUR-S.* This set of four data placement strategies are from Section 6, where data items exceeding the maximal storage limit at the assigned server by OUR-B will be offloaded to nearby servers. Since we have two

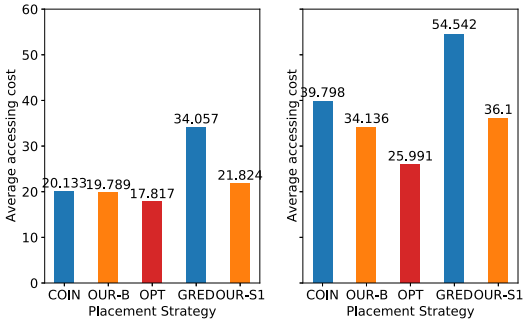
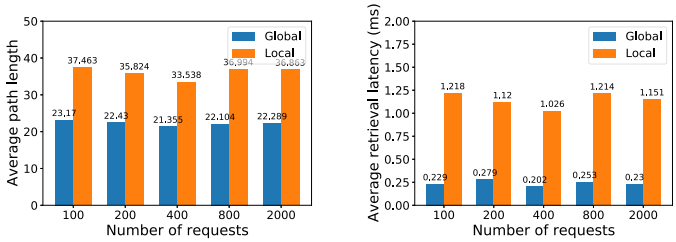


Fig. 11. Comparison with existing data placement methods [31], [32]. Left: data requests are randomly generated at random server/location. Right: data requests are randomly generated at random servers at the edge of the network.



(a) average path length (b) average retrieve latency

Fig. 12. Comparison of global retrieve and local retrieve in OUR-B with different numbers of data requests.

methods for processing offloading in different orders (Algorithms 1 and 2) and two methods for different offloading choices (*Nearest_Neighbor* and *Nearest_Server*), we have four different OUR-S methods in total:

- *OUR-S1*: Algorithm 1 + Nearest_Neighbor,
- *OUR-S2*: Algorithm 1 + Nearest_Server,
- *OUR-S3*: Algorithm 2 + Nearest_Neighbor,
- *OUR-S4*: Algorithm 2 + Nearest_Server.

- *OUR-R*. This is the data placement strategy from Section 7, which places multiple replicas in the network and the number of replicas $n(d_i)$ of a data item d_i depends its popularity, size and available storage. For comparison, we also implement and test another version *OUR-R-fixed* where fixed numbers of replicas are used.

For all of these data placement strategies, both global retrieve and local retrieve can be used. At last, the simulation runs 100 times to get the average result.

8.2 Comparison With Existing Methods

First, we compare the performance of our proposed data placement methods with existing methods from [31] and [32]. Recall that both COIN [31] and GRED [32] also use virtual space based methods to place data indexes or items but did not consider the data popularity, and their work inspire our proposed methods. Compared with COIN, GRED uses the centrodial Voronoi tesselation to handle load balance among edge servers. We also implement an optimal strategy (OPT) where the data item is placed to the edge server that has the overall shortest path length to all other edge servers. In this set of simulations, we compare our methods OUR-B and OUR-S1 to COIN, GRED, and OPT with the same setting. The number of data requests is set to 1,500 and global retrieve is applied. Furthermore, in this subsection, we only compare methods without replica since methods with replica always have better performance. The results are reported in Fig. 11.

The left plot shows the average path length when the data requests come from random edge servers. Clearly, when the data storage limit at each server is not considered, our proposed basic method OUR-B preforms similar to COIN with a slightly shorter average path length and is close to the OPT. But when we consider the storage limit, our method OUR-S1 can significantly reduce the path length compared with GRED. This confirms that the advantage of taking data popularity into consideration during both data placement and offloading. Compared with OUR-B, OUR-S1 only increases the average path length a little (while GRED's path is much longer than COIN). Recall that offloading data to other servers will increase the retrieve paths.

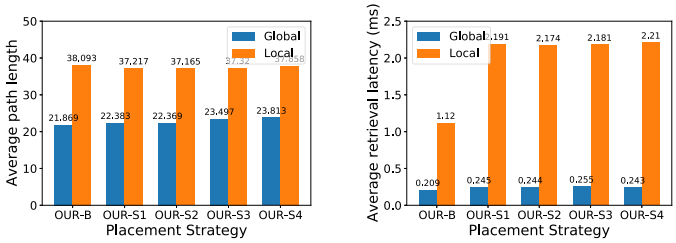
The right plot shows the results when the data requests come from the edge servers near the boundary of the network. Since these requests need longer path to reach other parts of the network, the path lengths of all methods are longer than those in left plot. In addition, we can also observe that our proposed algorithms (OUR-B or OUR-S1) are much better than the existing methods (COIN or GRED) in this case. This is mainly due to that our proposed methods consider data popularity and ensure that the more popular data is closer to the network center where the average path length to boundary region is much shorter.

Obviously, without considering data storage limit, the average accessing cost of OPT is better than all other methods. However, such a optimal method will increase the storage burden of the selected server, because all data items are placed on the single optimal server. In contrast, our proposed methods spread all data to different servers based on their data popularity and virtual distances. Thus, our methods can balance the storage burden among edge servers while keeping relevantly small accessing cost as shown in Fig. 11. In addition, if the storage limit and/or request distribution is considered, finding the optimal server becomes a very challenging optimization problem.

8.3 Global Retrieve Versus Local Retrieve

We now compare two retrieve strategies: global retrieve and local retrieve. First, we use OUR-B as the placement strategy, and vary the number of total requests from 100 to 2,000. Fig. 12a shows the average length of path travelled by data requests. Note that the average path length of different number of requests is almost the same for both retrieve methods. This is reasonable since the data placement is static. More importantly, global retrieve strategy has a shorter average path length than local retrieve strategy. This is due to that the global strategy always takes the shortest path between source and target servers. Fig. 12b presents the average retrieve latency of two retrieve methods. As we can see, the average retrieve latency of local strategy is far larger than that of global strategy. This is due to that (1) local retrieve strategy takes more number of hops during the forwarding; (2) it also may need to perform random forwarding to escape from local minimums.

Second, we fixed the number of requests at 1,000 and test the two retrieve strategies with both OUR-B and four OUR-



(a) average path length (b) average retrieve latency

Fig. 13. Comparison of global retrieve and local retrieve in OUR-B and OUR-S with 1,000 data requests.

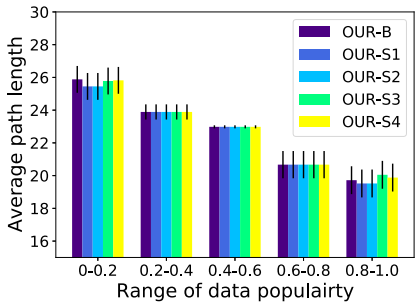


Fig. 14. Data popularity versus path length.

S placement methods. Fig. 13 present the results. From the results, we can draw the following conclusions. (1) Similar to results in Fig. 12, local retrieve takes longer path and latency than global retrieve does in all cases. (2) For the average path length in global retrieve, it is clear that those of OUR-S are longer than that of OUR-B. Remember that in OUR-S data items may be offloaded to another edge server rather than the nearest server to the data. Thus, global retrieve needs an additional path to reach the target server. (3) For the local retrieve, the average path lengths of OUR-S are shorter than that of OUR-B. This might be due to that some of data items are offloaded to servers which are closer to the request edge server. (4) It is obvious that the average retrieve latency of OUR-S is longer than that of OUR-B for both global and local retrieve since the retrieve takes additional time to find where the data is stored. Interestingly, even though the average path length of local retrieve for OUR-S is shorter than that for OUR-B, local retrieve still needs more time to figure out the location of the data item thus lead to a much longer retrieve latency than OUR-B.

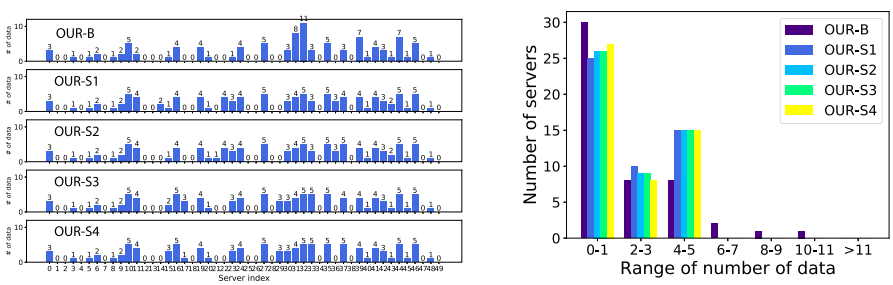
In summary, local strategy in general leads to longer average path length and larger retrieve latency than global retrieve. However, local strategy only utilizes the neighbors' information to compute the forwarding decision, which saves the storage of all shortest paths and makes it work well in scale. Therefore, there is a trade-off between the performance (retrieve latency) and the complexity (computing, storing and updating the global shortest paths). For all simulation results in the remaining section, we only report the results from global retrieve due to space limit.

8.4 Path Length Versus Data Popularity

One of the unique designs of proposed placement methods is that the placement of a data item depends on its data popularity. In this set of simulations, we study the relationship between data popularity and average path length by taking a close look at the path length for each data item. We ensure that each edge server requests the same set of 100 data items, and then measure the average path length of each method for each data item. We expect that the more popular is the data, the shorter is the average path length. Results in Fig. 15 confirm our expectation. In the figure, we group the data items into five groups based on their normalized popularity (ranging from 0 to 1) and then report the average path length of each group. Clearly, the overall trend of average path length decreases when data popularity increases. This proves the advantage of taking data popularity consideration in data placement. The performances of OUR-B and OUR-S are not much different, since the offloading does not happen a lot in this simulation.

8.5 Placement Strategies with Storage Limits

In this set of simulations, we aims to illustrate the power of offloading in data placement when we consider the maximal storage limits. Fig. 15 shows the distribution of number of data items placed at each server for both OUR-B and OUR-S. The top subplot in Fig. 15a presents the detailed load of OUR-B at each server which ignores the storage limit. Clearly, the edge server 32 is overloaded with data items. The other four subplots are results from four OUR-S, obviously they can spread the overloaded data items to other edge servers to make load more evenly. Fig. 15b also shows the aggregated distribution of number of server over the number of placed data items. The same conclusion can be drawn that OUR-S reduces number of servers with high loads (such as those with more than 6 data items in OUR-B). These results confirm the advantage of proposed offloading strategies in Section 6.



(a) loads among all servers (b) server load distribution

Fig. 15. Distribution of placed data items among servers for OUR-B and four OUR-S strategies.

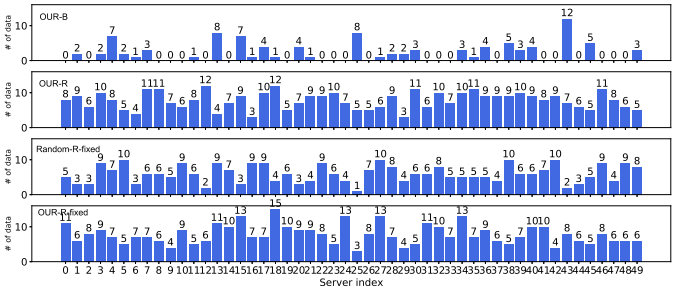
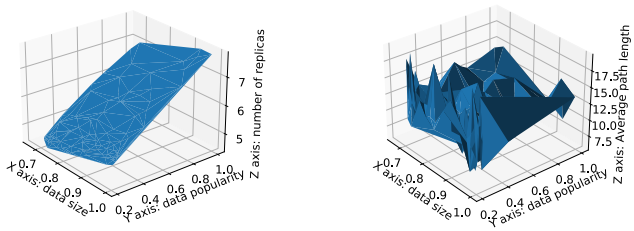


Fig. 16. Loads among servers with multiple data replicas.



(a) number of replicas (b) average path length

Fig. 18. Impact of data popularity and data size.

8.6 Placement Strategies With Data Replicas

Finally, we evaluate our proposed placement strategy OUR-R with data replication. Here, we test the following four data placement methods:

- *OUR-B*, the basic placement strategy where only single copy of data item is placed.
- *OUR-R*, the proposed placement strategy with multiple replicas, where the number of replicas is calculated based on data popularity, data size and available storage.
- *Random-R-fixed*, a random placement strategy with fixed number of replicas, where a fixed number of replicas of each item are randomly placed at edge servers.
- *OUR-R-fixed*, a variation of the proposed placement strategy with multiple replicas, where the number of replicas of each item is fixed.

To treat all methods with multiple replicas fairly, we let the fixed number of replicas in Random-R-fixed/OUR-R-fixed is equal to the average number of replicas on OUR-R.

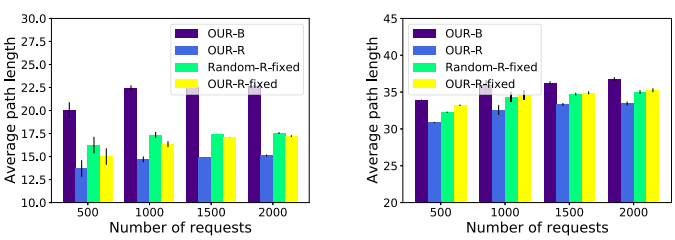
First, we display the loads among all servers with multiple data replicas as shown in Fig. 16. As we can see, there are more data in each edge server compared with the single replica strategy OUR-B. In addition, the difference of loads between these three multiple replica strategies is not obvious due to the data duplication. However, the average path length is different as shown in the next figure.

Fig. 17 shows the results of all methods under either global retrieve or local retrieve. First, the average path length of local retrieve is longer than that of global retrieve, which is the same conclusion from previous simulations. Second, all methods with multiple replicas perform much better than the single replica does. Compared to the single replica strategy, OUR-R reduces the average path length up to 36 percent. This confirms that data replication can significantly reduce the average path length of data request. Third, both with multiple replicas, OUR-R performs better than OUR-R-fixed, This shows the advance of using carefully designed number replica estimation with data popularity over fixed number replicas (evenly distributed among data items). Fourth, In global retrieve, OUR-R-fixed performs better than Random-R-fixed, which shows the proposed placement with Voronoi diagram is much better than random placement. However, OUR-R-fixed performs worse than Random-R-fixed in local retrieve since Random-R-fixed method may place multiple replicas near the request server randomly. In summary, among all methods with multiple replicas, our proposed OUR-R has the best results.

Fig. 18 plots the number of replicas $n(d_i)$ and average path length of each individual data item d_i for OUR-R in an instant of our simulation. It is clear that the number of replicas of d_i in OUR-R increases with its data size and data popularity as shown in Fig. 18a, which is due to Equation (4). Then larger number of replicas usually leads to smarter average path length due to our placement scheme, as shown in Fig. 18b.

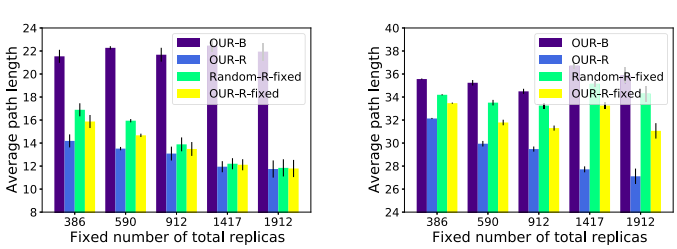
Fig. 19 shows the results of all methods with 1,000 requests when we increase the total number of data replicas. Obviously, with more data replicas, the shorter the average path length is. In addition, for global retrieve, when the number of data replicas is large, the difference among all replication placements becomes smaller. This is reasonable since with such large number of replicas, the placement does not matter anymore. All of other conclusions from Fig. 17 are consistent here too.

In summary, our proposed data placement strategies (including offloading and replication methods) can reduce the shortest path and balance the load among edge servers by taking the advantage of considering data popularity.



(a) global retrieve (b) local retrieve

Fig. 17. Comparison of placement strategies with multiple replicas.



(a) global retrieve (b) local retrieve

Fig. 19. Impact of the number of data replicas.

9 Conclusion

Data placement has become a critical issue in edge computing where data is generated from edge units and stored within edge network. In this paper, we have studied data placement strategy for edge network which takes data popularity into consideration. Based on their data popularity, data items are mapped to a virtual plane where network topology of edge servers is embedded. The mapping method puts more popular data closer to the network center in order to shorten its retrieve path from all other regions. Corresponding placement and retrieve strategies can then be easily performed based on distance measurement between virtual coordinators of data items and edge servers. We also design several multiples offloading and replication methods to overcome the storage limits and further improve the performance. Simulation results confirm the effectiveness and efficiency of our proposed strategies.

This work is the first step towards more intelligent data placement in edge computing by demonstrating the power of considering data popularity. There are a few possible directions for further study. (1) We mainly consider a static data placement where placement decision is static. If the edge environment is dynamic, one simple way to address it is to update the virtual coordinates accordingly and then adjust the placement. If there is a new data item or an update of a data popularity, a new calculation of its coordinates in the virtual plane can be easily performed. If there is an update of network topology, the edge servers' coordinates need to be regenerated, which can also be efficiently handled by the controller of software-defined edge network infrastructure (Fig. 4). Compared with optimization based solutions, the proposed method is more agile. If more complex dynamics need to be considered, new and online placement strategies are needed. (2) We will also consider other data popularity metrics beyond the number of accesses (such as life-time) and more complex data popularity models, where various user or location specific preferences may be considered differently. One possible solution is treating the data preferences from different users/locations with different weights. Note that in such cases the optimization based solution might become too challenging to be solvable. (3) Depending on what type of computing tasks is applied to placed data items, task scheduling can be jointly optimized with data placement problem in the edge computing paradigm. This becomes more challenging but interesting. We leave studying of these issues as our future work.

Acknowledgments

This work was partially supported by the National Science Foundation under Grants CCF-1908843 and CNS-2006604.

References

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

[2] Y. Huang, J. Zhang, J. Duan, B. Xiao, F. Ye, and Y. Yang, "Resource allocation and consensus on edge blockchain in pervasive edge computing environments," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 1476–1486.

[3] Y. Wang, Y. Dong, S. Guo, Y. Yang, and X. Liao, "Latency-aware adaptive video summarization for mobile edge clouds," *IEEE Trans. Multimedia*, vol. 22, no. 5, pp. 1193–1207, May 2020.

[4] S. Yang *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, Jan. 2021.

[5] R. Bi, Q. Liu, J. Ren, and G. Tan, "Utility aware offloading for mobile-edge computing," *Tsinghua Sci. Technol.*, vol. 26, no. 2, pp. 239–250, 2020.

[6] Q. Meng, K. Wang, X. He, and M. Guo, "QoE-driven big data management in pervasive edge computing environment," *Big Data Mining Anal.*, vol. 1, no. 3, pp. 222–233, 2018.

[7] S. Nath and J. Wu, "Deep reinforcement learning for dynamic computation offloading and resource allocation in cache-assisted mobile edge computing systems," *Intell. Converged Netw.*, vol. 1, no. 2, pp. 181–198, 2020.

[8] J. Zhang, B. Chen, Y. Zhao, X. Cheng, and F. Hu, "Data security and privacy-preserving in edge computing paradigm: Survey and open issues," *IEEE Access*, vol. 6, pp. 18209–18237, 2018.

[9] Y. Liu, T. Feng, M. Peng, J. Guan, and Y. Wang, "DREAM: Online control mechanisms for data aggregation error minimization in privacy-preserving crowdsensing," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2020.3011679.

[10] T. Li *et al.*, "Privacy-preserving participant grouping for mobile social sensing over edge clouds," *IEEE Trans. Netw. Sci. Eng.* vol. 8, no. 2, pp. 865–880, Second Quarter 2021.

[11] P. Chundi, D. J. Rosenkrantz, and S. S. Ravi, "Deferred updates and data placement in distributed databases," in *Proc. IEEE 12th Int. Conf. Data Eng.*, 1996, pp. 469–476.

[12] H. I. Abdalla, "An efficient approach for data placement in distributed systems," in *Proc. 5th IEEE/FTRA Int. Conf. Multimedia Ubiquitous Eng.*, 2011, pp. 297–301.

[13] A. Brinkmann, K. Salzwedel, and C. Scheideler, "Efficient, distributed data placement strategies for storage area networks," in *Proc. 12th ACM Symp. Parallel Algorithms Architectures*, 2000, pp. 119–128.

[14] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *Proc. 16th Int. Conf. Supercomputing*, 2002, pp. 84–95.

[15] M. Tu, H. Ma, L. Xiao, I.-L. Yen, F. Bastani, and D. Xu, "Data placement in P2P data grids considering the availability, security, access performance and load balancing," *J. Grid Comput.*, vol. 11, no. 1, pp. 103–127, 2013.

[16] J. Tian, Z. Yang, and Y. Dai, "A data placement scheme with time-related model for P2P storages," in *Proc. 7th IEEE Int. Conf. Peer-to-Peer Comput.*, 2007, pp. 151–158.

[17] K. Rzadca, A. Datta, and S. Buchegger, "Replica placement in P2P storage: Complexity and game theoretic analyses," in *Proc. IEEE 30th Int. Conf. Distrib. Comput. Syst.*, 2010, pp. 599–609.

[18] M. Drwal and J. Józefczyk, "Decentralized approximation algorithm for data placement problem in content delivery networks," in *Proc. Doctoral Conf. Comput., Elect. Ind. Syst.*, 2012, pp. 85–92.

[19] M. H. Al-Shayeji, S. Rajesh, M. Alsarraf, and R. Alsuwaid, "A comparative study on replica placement algorithms for content delivery networks," in *Proc. 2nd Int. Conf. Adv. Comput., Control, Telecommun. Technol.*, 2010, pp. 140–142.

[20] Q. Li, K. Wang, S. Wei, X. Han, L. Xu, and M. Gao, "A data placement strategy based on clustering and consistent hashing algorithm in cloud computing," in *Proc. 9th Int. Conf. Commun. Netw. China*, 2014, pp. 478–483.

[21] Q. Xu, Z. Xu, T. Wang, "A data-placement strategy based on genetic algorithm in cloud computing," *Int. J. Intell. Sci.*, vol. 5, no. 3, 2015, Art. no. 145.

[22] W. Guo and X. Wang, "A data placement strategy based on genetic algorithm in cloud computing platform," in *Proc. IEEE 10th Web Inf. Syst. Appl. Conf.*, 2013, pp. 369–372.

[23] T. Wang, S. Yao, Z. Xu, and S. Jia, "DCCP: An effective data placement strategy for data-intensive computations in distributed cloud computing systems," *J. Supercomputing*, vol. 72, no. 7, pp. 2537–2564, 2016.

[24] Q. Zhao, C. Xiong, X. Zhao, C. Yu, and J. Xiao, "A data placement strategy for data-intensive scientific workflows in cloud," in *Proc IEEE/ACM 15th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2015, pp. 928–934.

[25] M. Wang, J. Zhang, F. Dong, and J. Luo, "Data placement and task scheduling optimization for data intensive scientific workflow in multiple data centers environment," in *Proc. IEEE Int. Conf. Adv. Cloud Big Data*, 2014, pp. 77–84.

[26] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A data placement strategy in scientific cloud workflows," *Future Gener. Comput. Syst.*, vol. 26, no. 8, pp. 1200–1214, 2010.

[27] Y. Shao, C. Li, and H. Tang, "A data replica placement strategy for iot workflows in collaborative edge and cloud environments," *Comput. Netw.*, vol. 148, pp. 46–59, 2019.
[28] B. Lin *et al.*, "A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 4254–4265, Jul. 2019.
[29] C. Li, J. Bai, and J. Tang, "Joint optimization of data placement and scheduling for improving user experience in edge computing," *J. Parallel Distrib. Comput.*, vol. 125, pp. 93–105, 2019.
[30] M. Breitbach, D. Schäfer, J. Edinger, and C. Becker, "Context-aware data and task placement in edge computing environments," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2019, pp. 1–10.
[31] J. Xie, C. Qian, D. Guo, M. Wang, S. Shi, and H. Chen, "Efficient indexing mechanism for unstructured data sharing systems in edge computing," in *Proc. IEEE IEEE Conf. Comput. Commun.*, 2019, pp. 820–828.
[32] J. Xie, C. Qian, D. Guo, X. Li, S. Shi, and H. Chen, "Efficient data placement and retrieval services in edge computing," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 1029–1039.
[33] Y. Huang, X. Song, F. Ye, Y. Yang, and X. Li, "Fair and efficient caching algorithms and strategies for peer data sharing in pervasive edge computing environments," *IEEE Trans. Mobile Comput.*, vol. 19, no. 4, pp. 852–864, Apr. 2019.
[34] G. Sallam and B. Ji, "Joint placement and allocation of virtual network functions with budget and capacity constraints," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 523–531.
[35] S. Yang, F. Li, S. Trajanovski, X. Chen, Y. Wang, and X. Fu, "Delay-aware virtual network function placement and routing in edge clouds," *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 445–459, Feb. 2021.
[36] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 514–522.
[37] V. Farhadi *et al.*, "Service placement and request scheduling for data-intensive applications in edge clouds," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1279–1287.
[38] L. Zhao, W. Sun, Y. Shi, and J. Liu, "Optimal placement of cloudlets for access delay minimization in SDN-based internet of things networks," *IEEE Internet of Things J.*, vol. 5, no. 2, pp. 1334–1344, Apr. 2018.
[39] L. Zhao and J. Liu, "Optimal placement of virtual machines for supporting multiple applications in mobile edge networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 7, pp. 6533–6545, Jul. 2018.
[40] S. Yang, F. Li, M. Shen, X. Chen, X. Fu, and Y. Wang, "Cloudlet placement and task allocation in mobile edge computing," *IEEE Internet of Things J.*, vol. 6, no. 3, pp. 5853–5863, Jun. 2019.
[41] L.-W. Lee, P. Scheuermann, and R. Vingralek, "File assignment in parallel I/O systems with minimal variance of service time," *IEEE Trans. Comput.*, vol. 49, no. 2, pp. 127–140, Feb. 2000.
[42] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng, "CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2010, pp. 188–196.
[43] J. MacCormick, N. Murphy, V. Ramasubramanian, U. Wieder, J. Yang, and L. Zhou, "Kinesis: A new approach to replica placement in distributed storage systems," *ACM Trans. Storage*, vol. 4, no. 4, pp. 1–28, 2009.
[44] T. Janaszka, D. Bursztynowski, and M. Dzida, "On popularity-based load balancing in content networks," in *Proc. IEEE 24th Int. Teletraffic Congr.*, 2012, pp. 1–8.
[45] J. Li *et al.*, "Popularity-driven coordinated caching in named data networking," in *Proc. ACM/IEEE Symp. Architectures Netw. Commun. Syst.*, 2012, pp. 15–26.
[46] C. Hamdeni, T. Hamrouni, and F. B. Charrada, "Data popularity measurements in distributed systems: Survey and design directions," *J. Netw. Comput. Appl.*, vol. 72, pp. 150–161, 2016.
[47] Y. Wang, C.-W. Yi, M. Huang, and F. Li, "Three dimensional greedy routing in large-scale random wireless sensor networks," *Ad Hoc Netw. J.*, vol. 11, no. 4, pp. 1331–1344, 2013.
[48] Y. Wang and X.-Y. Li, "Efficient Delaunay-based localized routing for wireless sensor networks," *Wiley Int. J. Commun. Syst.*, vol. 20, no. 7, pp. 767–789, 2007.
[49] S. S. Lam and C. Qian, "Geographic routing in d-dimensional spaces with guaranteed delivery and low stretch," *IEEE/ACM Trans. Netw.*, vol. 21, no. 2, pp. 663–677, Apr. 2013.
[50] J. Zhang, H. Guo, J. Liu, and Y. Zhang, "Task offloading in vehicular edge computing networks: A load-balancing solution," *IEEE Trans. Veh. Technol*, vol. 69, no. 2, pp. 2092–2104, Feb. 2020.
[51] C. S. M. Babou *et al.*, "Hierarchical load balancing and clustering technique for home edge computing," *IEEE Access*, vol. 8, pp. 127593–127607, 2020.
[52] L. Yang, H. Yao, J. Wang, C. Jiang, A. Benslimane, and Y. Liu, "Multi-UAV-enabled load-balance mobile-edge computing for IoT networks," *IEEE Internet of Things J.*, vol. 7, no. 8, pp. 6898–6908, Aug. 2020.
[53] H. Guo, J. Liu, and J. Lv, "Toward intelligent task offloading at the edge," *IEEE Netw.*, vol. 34, no. 2, pp. 128–134, Mar./Apr. 2019.
[54] News popularity in multiple social media platforms data set. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/News+Popularity+in+Multiple+Social+Media+Platforms

Xinliang Wei (Student Member, IEEE) received the BE and MS degrees in software engineering from SUN Yat-sen University, Guangzhou, China, in 2014 and 2016, respectively. He is currently working toward the PhD degree at the Department of Computer and Information Sciences, Temple University. His research interests include edge computing, Internet of Things, and computer graphs.

Yu Wang (Fellow, IEEE) received the BEng and MEng degrees in computer science from Tsinghua University, and the PhD degree in computer science from the Illinois Institute of Technology. He is a professor with the Department of Computer and Information Sciences, Temple University. His research interest includes wireless networks, smart sensing, and mobile computing. He has published more than 200 papers in peer reviewed journals and conferences. He is a recipient of Ralph E. Powe Junior Faculty Enhancement awards from Oak Ridge Associated Universities (2006), Outstanding Faculty Research Award from College of Computing and Informatics at the University of North Carolina at Charlotte (2008), and ACM Distinguished Member (2020).