Scalable and Adaptive Data Replica Placement for Geo-Distributed Cloud Storages

Kaiyang Liu, *Member, IEEE*, Jun Peng, *Member, IEEE*, Jingrong Wang, *Student Member, IEEE*, Weirong Liu, *Member, IEEE*, Zhiwu Huang, *Member, IEEE*, and Jianping Pan, *Senior Member, IEEE*

Abstract—In geo-distributed cloud storage systems, data replication has been widely used to serve the ever more users around the world for high data reliability and availability. How to optimize the data replica placement has become one of the fundamental problems to reduce the inter-node traffic and the system overhead of accessing associated data items. In the big data era, traditional solutions may face the challenges of long running time and large overheads to handle the increasing scale of data items with time-varying user requests. Therefore, novel offline community discovery and online community adjustment schemes are proposed to solve the replica placement problem in a scalable and adaptive way. The offline scheme can find a replica placement solution based on the average read/write rates for a certain period of time. The scalability can be achieved as 1) the computation complexity is linear to the amount of data items and 2) the data-node communities can evolve in parallel for a distributed replica placement. Furthermore, the online scheme is adaptive to handle the bursty data requests, without the need to completely override the existing replica placement. Driven by real-world data traces, extensive performance evaluations demonstrate the effectiveness of our design to handle large-scale datasets.

Index Terms—Geo-distributed storage system, data replica placement, scalability, adaptivity, community discovery

1 Introduction

In the current era of big data, geo-distributed cloud storage systems need to manage, manipulate, and analyze a large scale of data for the emerging data-intensive applications. According to the IDC report, the volume of data is doubling every two years and thus will reach a staggering 44 zettabytes by 2020 [1]. To serve the ever more users around the world, data replication among geo-distributed storages has been widely used to increase data reliability and availability [2].

Placing requested data closer to end users helps to lower the user experienced service delay and the inter-node data read traffic, which motivates intensive research about data replica placement. Modern service providers, e.g., Facebook, maintain a full copy of user data in each data center [3]. However, this may generate unnecessarily high inter-node synchronization traffic to maintain consistency among data and replicas. Therefore, the inter-node traffic can be reduced by selecting a proper number of data replicas.

Apart from the inter-node traffic, the storage locations of data replicas may also affect the system overhead of accessing associated data items [4], [5]. It is worth noting that users may request multiple data items in one transaction. For example, in online analytical processing (OLAP) systems, a query may be executed by accessing multiple data blocks [6]. The system overhead could be reduced if fewer storage nodes are involved to handle such a request. The reason is that a certain overhead, e.g., the establishment of TCP connections, will be introduced if the read request is dispatched to a storage node. In short, data replica placement reduces the system overhead by placing associated data items together in the same storage location. With the increasing number of data items, how to choose the proper number and storage locations of data replicas becomes a critical issue.

Various data replica placement schemes have been proposed to seek optimal data storage locations, which are typically implemented in a centralized/offline way: At every distributed storage node handling the user requests, the data access logs are captured. Then, a central controller is deployed to collect all logs and analyze the request frequency of each data item. The extracted information is fed into the replica placement algorithms, e.g., mathematical programming [8] and graph partitioning [5], [7], [9], which finally output the storage locations of data replicas. These centralized/offline schemes can iteratively approximate the optimal solutions with high accuracy.

- *K. Liu is with the School of Computer Science and Engineering, Central South University, Changsha 410075, China, and also with the Department of Computer Science, University of Victoria, Victoria, BC V8P 5C2, Canada. E-mail: liukaiyang@csu.edu.cn.*
- *J. Peng and W. Liu are with the School of Computer Science and Engineering, Central South University, Changsha 410075, China. E-mail: {pengj, frat}@csu.edu.cn.*
- *J. Wang is with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 1A1, Canada. E-mail: jr.wang@mail.utoronto.ca.*
- *Z. Huang is with the School of Automation, Central South University, Changsha 410075, China. E-mail: hzw@csu.edu.cn.*
- *J. Pan is with the Department of Computer Science, University of Victoria, Victoria, BC V8P 5C2, Canada. E-mail: pan@uvic.ca.*

Manuscript received 6 Mar. 2019; revised 5 Jan. 2020; accepted 15 Jan. 2020. Date of publication 21 Jan. 2020; date of current version 25 Feb. 2020. (Corresponding author: Zhiwu Huang.) Recommended for acceptance by H. Huang. Digital Object Identifier no. 10.1109/TPDS.2020.2968321

Although intuitively valid in design, the centralized/offline schemes may meet two practical challenges when applied to a large-scale storage system. First of all, a long running time of the placement scheme is expected when a large amount of data items are deployed at many storage nodes [4]. Furthermore, faced with time-varying data requests, these offline solutions are slow to react to the real-time changes in workloads [10]. For a large scale storage system with user request uncertainties, data replica placement schemes should be 1) highly efficient with small computation overhead for a quick placement decision, and 2) flexible to change the storage locations of data replicas in an online fashion. Therefore, it is imperative to solve the replica placement problem in a more scalable and adaptive way.

In this paper, based on the overlapping community discovery and adjustment, we design scalable and adaptive data replica placement schemes in geo-distributed cloud storage systems. A data-node community is defined as the group of a storage node and all data items placed at it, which should have more internal data access requests than external ones. Therefore, a more compact community structure means more data requests are served locally with lower system overhead and less inter-node traffic. Unlike traditional centralized placement schemes, communities can evolve to decide whether each data replica should be placed at the node in a parallel and adaptive way. The scalability of our design can be achieved by this distributed implementation along with the computation complexity linear to the amount of data items. Our major contributions in this paper include:

- A novel distributed overlapping community discovery scheme is proposed to solve the data replica placement problem in a scalable way. This offline scheme can find a replica placement solution based on the average read/write rates for a certain time period.
- Guided by the offline scheme, an online community adjustment scheme is proposed to adaptively handle the bursty requests.
- The worst-case performance guarantees of the proposed schemes are provided via theoretical analysis.
- Extensive evaluation results driven by real-world data traces show the superiority of our design over the state-of-the-art replica placement methods.

The rest of this paper is organized as follows. Section 2 summarizes the related work. Section 3 presents the model of the geo-distributed storage system. In Section 4, the distributed community discovery-based data replica placement scheme is proposed along with the theoretical analysis. In Section 5, an online community adjustment scheme is further proposed to handle bursty requests. In Section 6, the efficiency and the performance of the proposed schemes are evaluated and substantiated with extensive experiments. Section 7 concludes this paper.

2 Related Work

2.1 Data Replica Placement

Many researchers have pointed out that data replica improves the data locality to ensure a better read/write performance in data-intensive systems. In online social networks (OSN), Tran *et al.* [11] proposed to create data replicas for users only at the node having the largest amount of their friends to alleviate excessive data replication. However, only the read requests are considered, which may generate high data-write traffic. To minimize the total inter-node traffic, Liu *et al.* [3] proposed to select data items that have higher read rates and lower update rates for replication. Traverso *et al.* [12] proposed a lazy content update mechanism to lower the peak-hour traffic in the geo-replication system. All these previous studies only focus on the inter-node traffic or data access delay.

The system overhead is also affected by other factors, e.g., the multi-get hole effect [13], which could be explained as follows: When more than one items are requested in one data transaction, the span of involved nodes to handle such a data request influences the throughput of the storage system. Therefore, how to place strongly associated data items has attracted extensive research attention. Nishtala *et al.* [14] pointed out that data items that are frequently requested together can be treated as a whole, and such request pattern information can be extracted by analyzing historical traces. Agarwal *et al.* [4] presented Volley, an offline data placement framework for geo-distributed services where a data item is iteratively migrated to get near to both end users and its associated data items. Jiao *et al.* [9] proposed a multi-objective data placement (MODP) scheme, optimizing the data storage locations through iterative graph cuts. Yu *et al.* [5] designed a hypergraph-based framework for associated data placement (ADP) among geo-distributed storage nodes. Nevertheless, these centralized solutions are not effective enough, in terms of the running time and computation overhead, to output decisions quickly, especially when the storage system is on a large scale.

Therefore, it is critical to solving the replica placement problem in a more scalable way. Yu *et al.* [7] proposed a sketch-based data placement (SDP) for hypergraph sparsification, reducing the algorithm running time. SpeCH [15] used a randomized solution for the low-rank approximation of the hypergraph matrix, improving the efficiency of data placement. GPlacer [16] proposed heuristic solutions that can efficiently find sub-optimal data storage locations. However, these offline solutions are slow to react to the real-time changes in workloads, e.g., the bursty data requests.

As an online scheme, DataBot [17], [18] utilized reinforcement learning to adaptively learn optimal data placement policies, reducing the data access latency with no future assumption about the data requests. However, the inter-node traffic and the system overhead of accessing associated data items are overlooked in DataBot. Charapko *et al.* [19] proposed a data migration solution Akkio, which adapts to the changing access locality patterns. To improve the scalability of the solution for Petabytes of data at Facebook, Akkio grouped the related data with similar access locality into a migration unit. Akkio mainly focuses on the granularity of data migration, without considering the optimality of the solution. Table 1 compares the proposed schemes with existing research work. Different from previous studies, the proposed scheme in this paper takes the read/write requests and data association into account, and uses the community discovery/adjustment for scalable and adaptive data replica placement.

TABLE 1
Comparison of the Proposed Data Replica Placement Schemes With the Existing Research Work

Scheme	Performance Metrics	Solution	Efficiency	Adaptivity	Scalability
Volley [4]	Read/write traffic, latency, load balance	Iterative optimization	Low	Offline	Centralized
MODP [9]	Read/write traffic, operation cost	Graph cuts			
ADP [5]	Read traffic, overhead for associated data	Hypergraph partitioning			
SDP [7]		Hypergraph sparsification	Medium		
SpeCH [15]		Spectral clustering			
GPlacer [16]	Latency	Heuristic	High		
DataBot [17], [18]		Reinforcement learning		Online	
Akkio [19]	Read/write traffic, latency, storage footprint	Heuristic			
SD3 [3]	Read/write traffic		Very high		Distributed
Our schemes	Read/write traffic, overhead for associated data	Community discovery/ adjustment		Offline/online	

2.2 Community Discovery-Based Schemes

Community discovery is considered as an efficient solution to extract useful information from complex networks, which has been widely used in a series of domains, e.g., biological networks, social sciences, and regional geography [20]. Various methods have been proposed for detecting both non-overlapping [21], [22] and overlapping communities [23], [24]. For the data placement problem, Chen *et al.* [25] leveraged the interaction locality to merge the pair of communities into a single community, aiming to place the data items within an interaction community together. Without considering data replication, the performance gain is limited in this work. To achieve a perfect data locality in OSN, Pujol *et al.* [26] tried to place social communities at the same storage location and designed a partitioning scheme to replicate the data of all friends, which inevitably incurs a huge data synchronization traffic. Hu *et al.* [27] proposed a two-phase community discovery scheme to place associated data items to the cloud, speeding up the parallel processing stage of data analytics. Unlike the existing community discovery-based replica placement methods, our design tries to find a proper number of data replicas in each community, minimizing both the inter-node traffic and multi-get hole effect.

TABLE 2
Notations

Symbol	Definition
$\mathcal{N}$	Set of geo-distributed storage nodes, and $N = \lvert\mathcal{N}\rvert$
$\mathcal{M}$	Set of data items, and $M = \lvert\mathcal{M}\rvert$
ε_{xy}	Binary variable: To place a replica of data item x at node y ($\varepsilon_{xy} = 1$) or not ($\varepsilon_{xy} = 0$), $x \in \mathcal{M}$, $y \in \mathcal{N}$
p	Request pattern which involves multiple data items in one read transaction, $p \in \mathcal{P}$
δ_{pyj}	Binary variable: Request p from node y is routed to node j ($\delta_{pyj} = 1$) or not ($\delta_{pyj} = 0$)
P_x	Subset of request patterns which contains data item x, $P_x \subset \mathcal{P}$
R_{py}	Read request rate of pattern p from source node y
R_{xy}	Read request rate of data item x from source node y
W_x	Write request rate of data item x
y_x, C_x	Master node, set of follower nodes for data item x
D_y	Set of stored data items and replicas at node y
λ	Constant overhead to fulfill a read request at the storage node
η	Weight to trade off the metrics of inter-node traffic and system overhead of accessing associated data items
L	Optimization objective value
T	Time length of the service period
ϕ	Pre-defined value for the online community adjustment scheme

3 System Model and Problem Statement

In this section, we start by introducing the model of the geo-distributed storage system and then discuss how to optimize the storage locations of data replicas. The major notations used in this paper are summarized in Table 2.

3.1 Geo-Distributed Storage System and Data Items

Fig. 1 illustrates the model of the geo-distributed cloud storage system, which consists of a set of storage nodes $\mathcal{N}$ distributed at different geographical locations (with size $N = \lvert\mathcal{N}\rvert$). Let $\mathcal{M}$ denote the set of data items stored in the system (with size $M = \lvert\mathcal{M}\rvert$). The data items could be files, tables or blocks in practice. Similar to the widely used Hadoop [28] and Cassandra storage system [29], it can be assumed that the data items are with a default size.

To improve data reliability, fault-tolerance, and accessibility, data items are often stored in a one-leader multi-follower manner. Initially, each data item $x \in \mathcal{M}$ is uploaded to or generated at a single master node $y_x \in \mathcal{N}$, denoted by

$$\mathcal{D} : x \rightarrow y_x. \tag{1}$$

Furthermore, data replicas are created and stored at the follower nodes. A binary variable ε_{xy} is introduced to decide whether a replica of data item x is placed at node y ($\varepsilon_{xy} = 1$) or not ($\varepsilon_{xy} = 0$). Then, the set of follower nodes for data item x is given by

$$C_x = \{y \in \mathcal{N} \,|\, \varepsilon_{xy} = 1\}. \tag{2}$$

It is worth noting that a storage node can act as a master node and a follower node simultaneously for different data items. Then, the set of all stored data items and replicas at node y is represented by

$$D_y = \{x \in \mathcal{M} \,|\, y = y_x \vee \varepsilon_{xy} = 1\}. \tag{3}$$

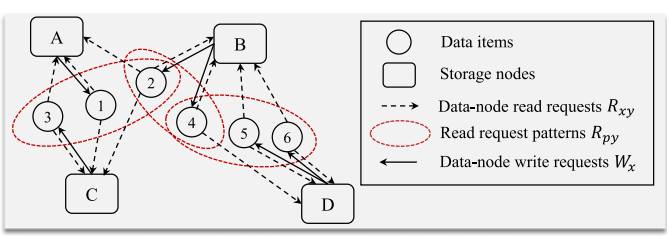


Fig. 1. Illustration of the geo-distributed storage system with the data read/write requests.

3.2 Workloads of Read/Write Requests

The data transactions contain a set of read and write requests. For the cloud-based storage system, the data access type could be both read-intensive and write-intensive [2]. Then, the models of read/write workloads are introduced.

3.2.1 Read Request Flow

The data-intensive applications, e.g., OLAP, may request multiple data items (denoted by *request pattern* p) in one transaction. Denote $\mathcal{P}$ the set of request patterns, $\mathcal{P} \subset \{\mathcal{M}, \emptyset\}$. Fig. 1 illustrates an example of the system which contains 6 data items and 3 request patterns, $\mathcal{P} = \{\{x_1, x_2, x_3\}, \{x_2, x_4\}, \{x_4, x_5, x_6\}\}$. Then, the data flow of read requests is defined as follows: When an application at a node requests a dataset p to fulfill its task, this node is set as the source of the flow, and the nodes having the data in p are the destinations. Furthermore, P_x is introduced to denote the subset of request patterns which contains data item x, $P_x \subset \mathcal{P}$. For example, $P_{x_2} = \{\{x_1, x_2, x_3\}, \{x_2, x_4\}\}$ in Fig. 1.

Let R_{py} denote the average read rate or frequency to the request pattern p from the source node y for a certain period T. Mature prediction methods, e.g., EWMA [30], could be applied to derive the request rates from the historical information, so the details of dealing with prediction are not included in this paper. The predicted read rate information R_{py} is fed into our scheme to make the replica placement decision. Then, the request rate R_{xy} for each data item x from node y can be calculated by

$$R_{xy} = \sum_{p \in \mathcal{P}} R_{py} \mathbf{1}(x \in p), \tag{4}$$

where the binary function $\mathbf{1}(x \in p)$ indicates whether the data x belongs to the request pattern p (denoted by $\mathbf{1}(x \in p) = 1$) or not ($\mathbf{1}(x \in p) = 0$).

3.2.2 Write Request Flows

For the reason of data security, whenever a data item x is to be written or updated, the write operation can only be submitted to the predefined master node y_x [2], [31]. Take Fig. 1 as an example, data x_1 can only be updated by the data owner at master node A. Then, for data consistency, the master node acts as the source to synchronize the updated data with all follower nodes. Inter-node write traffic will be generated to maintain data consistency. Let W_x denote the write rate to the data item x. Similar to R_{py}, W_x can also be derived from the historical information.

3.3 Performance Metrics

It has been pointed out that data placement can affect the performance of the storage system in both the inter-node traffic and the overhead of accessing associated data [5].

Inter-node traffic: For the data read requests, if the data item is not stored at the requesting node, the inter-node read traffic will be introduced. The total read traffic can be defined as

$$L^{[\mathrm{R}]} = \sum_{x \in \mathcal{M}} \sum_{y \in \mathcal{N}} R_{xy} \cdot [1 - \mathbf{1}(x \in D_y)], \tag{5}$$

where the binary variable $\mathbf{1}(x \in D_y)$ indicates whether item x is placed at node y or not. Furthermore, when the data item is written or updated, inter-node write traffic will be generated for synchronization. The total write traffic is defined as

$$L^{[\mathrm{W}]} = \sum_{x \in \mathcal{M}} W_x \cdot \sum_{y \in \mathcal{N}} \varepsilon_{xy}. \tag{6}$$

System overhead of accessing associated data: Let us consider the scenario that the associated data items in pattern p are requested from node y. Since the local node y might not be able to provide all needed data, all locally unobtainable data items should be fetched from remote nodes. As the routine process in handling a read request p may cause extra overheads, e.g., the establishment of TCP connections, the system overhead is related to the number of involved remote nodes [5], which can be defined as

$$\sum_{j \in \mathcal{N}, j \neq y} \lambda \cdot \delta_{pyj}, \tag{7}$$

where λ denotes the constant overhead to fulfill a read request at a remote node j, $j \neq y$. The binary variable $\delta_{pyj} \in \{0, 1\}$ indicates whether node j provides data access for request p or not. With different read rates to various request patterns, the total overhead to handle all the requests is given by

$$L^{[\mathrm{O}]} = \sum_{y \in \mathcal{N}} \sum_{p \in \mathcal{P}} R_{py} \cdot \sum_{j \in \mathcal{N}, j \neq y} \lambda \cdot \delta_{pyj}. \tag{8}$$

The operational cost of the cloud storage system can be reduced if (8) is minimized, which can be achieved by placing strongly associated data items together.

3.4 Optimization Problem Formulation

Our objective is to minimize the total service overhead by determining 1) where the data replicas should be placed ε_{xy}, and 2) where the read request should be routed δ_{pyj}

$$\begin{aligned} \min\ & L(\varepsilon_{xy}, \delta_{pyj}) = L^{[\mathrm{O}]} + \eta \cdot (L^{[\mathrm{R}]} + L^{[\mathrm{W}]}) \\ \text{s.t.}\ & \delta_{pyj} \leq \mathbf{1}(p \cap D_j), \\ & p \subset \bigcup\nolimits_{j = y \vee \delta_{pyj} = 1} D_j, \\ & \varepsilon_{xy}, \delta_{pyj} \in \{0, 1\}, \end{aligned} \tag{9}$$

where η is utilized to trade off the system overhead and inter-node traffic metrics defined above. Constraint $\delta_{pyj} \leq \mathbf{1}(p \cap D_j)$ ensures only nodes storing items in p can be set as the destinations of data requests. Constraint $p \subset \bigcup_{j = y \vee \delta_{pyj} = 1} D_j$ ensures the route destinations can provide all the needed data items. Furthermore, the storage capacity at each node is not considered as a constraint for the following two reasons: 1) From the user perspective, a cloud can provide "infinite" storage resources on demand; 2) Our work aims at a lower bound of the total service overhead by assuming an unlimited storage capacity at each node. The insights of the replica placement and request routing problem are as follows:

Insight 1. If more data replicas are placed at each node, the overhead $L^{[\mathrm{O}]}$ and inter-node read traffic $L^{[\mathrm{R}]}$ can be reduced. In the extreme case, if each node stores a full copy

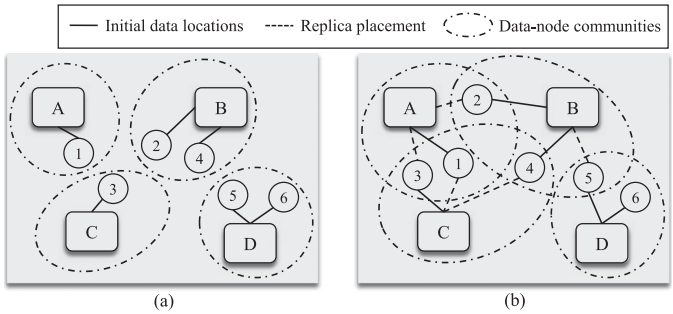


Fig. 2. An example of the overlapping community discovery-based approach: (a) Initial data-node community with data stored over the unique master node; (b) Local community expansion for replica placement over the follower nodes.

of all data items, $L^{[\mathrm{O}]}$ and $L^{[\mathrm{R}]}$ decrease to 0 as all read requests can be fulfilled locally. However, this will cause unnecessary write traffic for synchronization purposes.

Insight 2. It is worth noting that the formulated optimization problem is suitable for the scenario where data request traffic is fairly steady for the time period T. Therefore, the community discovery-based solution in Section 4 can find a replica placement solution based on the average read/write rates for this period. Furthermore, to handle the bursty requests, an online solution is proposed in Section 5.

Then, the hardness of the formulated optimization problem is examined. The replica placement and request routing decision is a 0-1 integer programming problem, which is proven NP-hard [32]. Furthermore, the difficulty is also partially due to the decision of replica placement and request routing affecting each other. In general, facing large amounts of data items, traditional solutions are less effective, in terms of the running time and overhead, to solve this problem.

4 Community Discovery Based Replica Placement

4.1 Design Overview

Considering large-scale datasets, a novel overlapping community discovery-based approach is proposed to solve the replica placement and request routing problem (9) in an efficient and scalable way. As shown in Fig. 2, a community is defined to include a set of data items and a storage node. Generally speaking, community discovery is to find tight-knit community structures that have more internal edges than external ones. As illustrated in Fig. 3, there are three kinds of external edges among the data-node communities:

(1) If a data item or its replica is not placed at a node, the data item needs to be fetched from another node when requested with a data-node read edge e_{xy}.
(2) The request pattern which involves multiple data items can be generalized as a hyperedge e_{py}.
(3) If a data replica is placed at a node, the data synchronization edge e_x will be introduced to maintain data consistency.

Then, it can be demonstrated that the weight of all external edges among communities is equivalent to the optimization objective value $L(\varepsilon_{xy}, \delta_{pyj})$ in (9) with the following theorem.

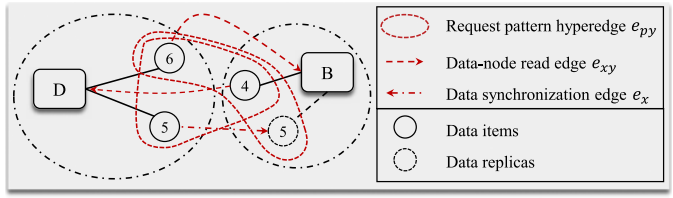


Fig. 3. Illustration of three kinds of external edges among communities: data-node read edge, request pattern hyperedge, and data synchronization edge.

Theorem 1. *If the weights of edges are set according to*

$$\begin{cases} w_{py} = \lambda R_{py}, \text{ for request pattern hyperedge } e_{py}, \\ w_{xy} = \eta R_{xy}, \text{ for data-node read edge } e_{xy}, \\ w_x = \eta W_x, \text{ for data synchronization edge } e_x, \end{cases} \tag{10}$$

the sum of external edge weights among communities is equivalent to the optimization objective value $L(\varepsilon_{xy}, \delta_{pyj})$.

Proof. Given the data-node community $\{y\} \cup D_y$, the number of read edges initiated from y is $\sum_{x \in \mathcal{M}} [1 - \mathbf{1}(x \in D_y)] \cdot \mathbf{1}(R_{xy} > 0)$. The number of write edges to node y is equal to the number of placed replicas $\sum_{x \in \mathcal{M}} \varepsilon_{xy}$. For the request pattern hyperedge, if the local community cannot provide all needed data items, the hyperedge may connect different communities, just as shown in Fig. 3. The number of external hyperedges from node y is $\sum_{p \in \mathcal{P}} \sum_{j \in \mathcal{N}, j \neq y} \delta_{pyj}$. With the edge weights in (10), the sum of external edge weights is

$$\begin{aligned} &\sum_{y \in \mathcal{N}} \sum_{x \in \mathcal{M}} \eta (R_{xy}[1 - \mathbf{1}(x \in D_y)] + W_x \varepsilon_{xy}) \\ &\quad + \sum_{y \in \mathcal{N}} \sum_{p \in \mathcal{P}} \sum_{j \in \mathcal{N}, j \neq y} \lambda R_{py} \delta_{pyj}, \end{aligned} \tag{11}$$

which is equal to $L(\varepsilon_{xy}, \delta_{pyj})$. □

This means that a compact community structure with fewer external edges can provide a better replica placement solution. The methodology of community discovery can be summarized as follows: Initially, each of the N storage nodes and its affiliated data items (according to $\mathcal{D}$ in (1)) are preassigned to a different community. Fig. 2a illustrates that the preassigned storage nodes and data items form N non-overlapping communities (also known as seeds). With the local expansion from seeds, data items are gradually added to the community, i.e., replicas are placed at storage nodes, based on some criteria, trying to minimize the weights of all external edges. As shown in Fig. 2b, the local expansion outputs the overlapping data-node communities with replicas.

4.2 Distributed Community Discovery Algorithm

Then, the design details of the distributed community discovery algorithm are presented to solve the replica placement and request routing problem. As the community discovery is based on the local expansion from seeds, we show how the external edges change when a data replica is added into a community first. Let us take Fig. 3 as an example with the initial data placement $\{x_4\} \rightarrow$ B and $\{x_5, x_6\} \rightarrow$ D. When data replica x_4 is placed at node D, two external edges, i.e., the request pattern hyperedge and the read

request edge to x_4, could become internal edges, and an external write edge should be created. However, when x_5 is added to B, the external hyperedge cannot be removed as B still needs to fetch x_6 from D. Therefore, when a data replica x is added at a node y, the weight variations of external edges are given by

$$v_{xy} = w_x - w_{xy} - \sum_{p \in P_x} w_{py} \theta_{xpy}, \tag{12}$$

where θ_{xpy} is a binary variable to decide whether the external hyperedge e_{py} can be removed ($\theta_{xpy} = 1$) or not ($\theta_{xpy} = 0$). Then, how θ_{xpy} can be derived from the current data locations D_y and request routing δ_{pyj} is shown. For data item x in pattern p requested by node y, the set of nodes which can provide x for R_{py} is given by

$$\{i \in \mathcal{N} \mid x \in D_i, \delta_{pyi} = 1\}. \tag{13}$$

Then, by assuming data replica x is placed at node y, θ_{xpy} can be derived from

$$\theta_{xpy} = \min\{1, \sum_{i \in \mathcal{N}, x \in D_i, \delta_{pyi}=1} \mathbf{1}(p \subset \bigcup\nolimits_{j \in N, \delta_{pyj}=1, j \neq i} D_j)\}, \tag{14}$$

where $\mathbf{1}(p \subset \bigcup_{j \in N, \delta_{pyj}=1, j \neq i} D_j)\} = 1$ indicates if node i is removed from the routing destination ($\delta_{pyi} \leftarrow 0$), other nodes ($\delta_{pyj} = 1$, $j \neq i$) can still provide all needed items for R_{py}. If at least one node i could be removed, the external hyperedge from y to i can be removed, and θ_{xpy} is set to 1.

Based on the analysis above, the criterion of community expansion is set as: If $v_{xy} \leq 0$, data replica x is added to the community by placing it at node y. This criterion ensures the efficiency of decision making facing a large amount of data items. The pseudo code of the distributed community discovery is listed in Algorithm 1. It is worth noting that in order to improve the scalability of our design, each data-node community evolves in parallel for decision making.

Algorithm 1. Distributed Community Discovery

Input: Dataset $\mathcal{M}$, node set $\mathcal{N}$, request pattern read rate R_{py}, data write rate W_x, master node y_x.
Output: Data replica placement ε_{xy}, request routing δ_{pyj}.
Initialization: $\forall \varepsilon_{xy} \leftarrow 0$, $\delta_{pyj} \leftarrow 1$ if $j = y_x$.
1: **for** Data item $x \in \mathcal{M}$, $x \notin D_y$ **do**
2: $\varepsilon_{xy} \leftarrow 1$, $D_y \leftarrow x$, if $R_{xy} \geq W_x$;
▷ Initial replica placement
3: **end for**
4: Exchange the data storage location information D_j with all other nodes, $j \in \mathcal{N}$;
5: **for** Request pattern $p \in \mathcal{P}$ **do**
6: Calculate request routing $\{\delta_{pyj}\}$ based on D_j, $j \in \mathcal{N}$;
7: **end for**
8: **for** Data item $x \in \mathcal{M}$, $x \notin D_y$ **do**
9: Calculate θ_{xpy} based on (14), $\forall p \in P_x$;
10: $\varepsilon_{xy} \leftarrow 1$, $D_y \leftarrow x$, if $v_{xy} \leq 0$;
▷ Community expansion
11: **end for**
12: Repeat Step 4 – 7 to update the request routing $\{\delta_{pyj}\}$ based on the D_j after the expansion, $j \in \mathcal{N}$

Initially, for each data item which satisfies $R_{xy} \geq W_x$, a data replica is directly created at node y ($\varepsilon_{xy} \leftarrow 1$, $D_y \leftarrow x$) as it can always benefit the weight reduction of external edges. In this step, the total inter-node traffic is minimized. Then, node y exchanges the storage location information D_j with all other nodes, $j \in \mathcal{N}$. The routing of all request patterns R_{py} from node y can be calculated based on D_j. As the amount of inter-node traffic is determined, the optimization problem in (9) for a read request R_{py} is transformed to

$$\begin{aligned} &\min \sum\nolimits_{j \in \mathcal{N}, j \neq y} \delta_{pyj} \text{s.t.} \quad \delta_{pyj} \leq \mathbf{1}(p \cap D_j), \\ &p \subset \bigcup\nolimits_{j=y \vee \delta_{pyj}=1} D_j, \qquad \delta_{pyj} \in \{0, 1\}. \end{aligned} \tag{15}$$

Therefore, the objective of routing is to find the minimum number of nodes to serve each read request, which can be classified into the classical NP-hard set cover problem [33]. To achieve a high solution efficiency, the well-studied greedy method is applied here. The storage node which holds the highest number of uncovered data items in the earlier iterations is greedily selected. The competitive ratio of this heuristic method is 2 with the computation complexity of $O(N^2)$ [33]. Based on the updated routing decision, the placement of data replica x will be decided by the expansion criterion. All the rest data items will be considered one-by-one to decide whether it should be added to the community. At last, the routing of all request patterns will be updated again to search for a better routing decision based on the current data-node community D_j, $j \in \mathcal{N}$.

4.3 Theoretical Analysis

The performance of the proposed distributed community discovery algorithm is theoretically analyzed in terms of the computation complexity and the worst-case performance bound.

Property 1. *Algorithm 1 has the computation complexity of* $O(|\mathcal{P}| \cdot N^2 + |P_x| \cdot N \cdot M)$.

Proof. If $R_{xy} \geq W_x$, the data items are directly added to the community with the computation complexity of $O(M)$ (Step 1–3). The exchange of the data storage location information with all other nodes needs N iterations (Step 4). Then, the routing for all request patterns $\mathcal{P}$ at node y should be calculated (Step 5–7). The used greedy heuristic method can solve the set cover problem in (15) in a polynomial time $O(N^2)$. Moreover, to calculate θ_{xpy} (Step 9), all nodes storing data item x should be considered with the maximum computation complexity $O(N)$. As data items may be requested in at most $|P_x|$ patterns at node y, it needs $|P_x| \cdot N \cdot M$ iterations to decide whether data items should be added to the community or not. Furthermore, the routing update process of all request patterns (Step 12) also needs $N + |\mathcal{P}| \cdot N^2$ iterations. To sum up, Algorithm 1 needs $M + 2(|\mathcal{P}| \cdot N^2 + N) + |P_x| \cdot N \cdot M$ iterations in total. The overall computation complexity is given by $O(|\mathcal{P}| \cdot N^2 + |P_x| \cdot N \cdot M)$. □

Theorem 2. *Algorithm 1 incurs no more than* $1 + \frac{\lambda}{\eta}$ *times of the optimal objective value* L_{opt}.

Proof. Let us consider an arbitrary data item x_1. If x_1 is always requested solely from y, i.e., $p = \{x_1\}$, the weight

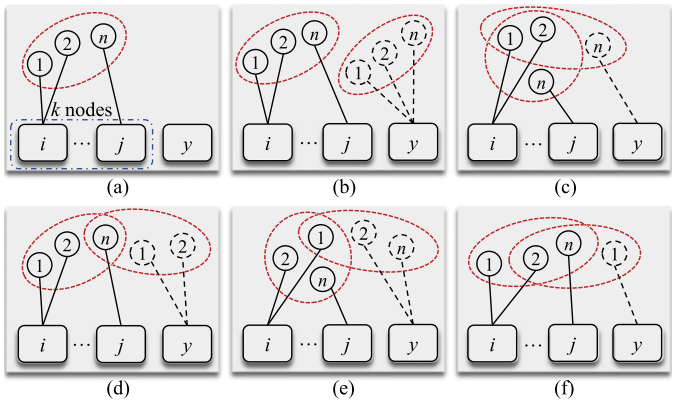


Fig. 4. Illustration of replica placement at node y when $y_{x_1} = y_{x_2} = i$ and $y_{x_n} = j$: (a) No replication; (b) Full data replication x_1, x_2 and x_n; (c) Partial data replication x_n; (d) Partial data replication x_1 and x_2; (e) Partial data replication x_2 and x_n; (f) Partial data replication x_1.

variations of external edges for the placement of x_1 at y are

$$v_{x_1y} = (W_{x_1} - R_{x_1y}) \cdot \eta - R_{py} \cdot \lambda. \tag{16}$$

In this case, Algorithm 1 can always find the optimal solution for x_1 based on the defined criterion of community expansion.

Then, let us consider the scenario when n data items are simultaneously requested from node y, $n \geq 2$. As shown in Fig. 4, data item x_1 and x_2 are assumed to be placed at a node i initially, i.e., $y_{x_1} = y_{x_2} = i$. This represents the case that multiple data items are stored at the same node. Furthermore, x_n is assumed to be placed at a node j, $y_{x_n} = j$, which represents another case that the requested data items are solely stored at each node. Given the storage locations of data items, it can be assumed that k storage nodes are involved to fulfill the data request, just as illustrated in Fig. 4a. According to (12), the weight variations of external edges are given by

$$\begin{cases} v_{x_1y} = (W_{x_1} - R_{x_1y}) \cdot \eta, \\ v_{x_2y} = (W_{x_2} - R_{x_2y}) \cdot \eta, \\ \cdots \\ v_{x_ny} = (W_{x_n} - R_{x_ny}) \cdot \eta - \sum_{p \in \mathcal{P}} R_{x_ny} \lambda \cdot \mathbf{1}(x_n \in p). \end{cases} \tag{17}$$

Then, we have the following six different cases:

$v_{x_1y}, v_{x_2y}, \ldots, v_{x_n}y > 0$: Based on Algorithm 1, no replica should be placed at node y, just as shown in Fig. 4a. Then, the weight of external edges is given by

$$\begin{aligned} L = k \cdot \sum_{p \in \mathcal{P}} R_{py} \lambda \cdot \mathbf{1}(x_1, x_2, \ldots, x_n \in p) \\ + (R_{x_1y} + R_{x_2y} + \cdots + R_{x_ny}) \cdot \eta. \end{aligned} \tag{18}$$

However, it is possible that data replica $x_1, x_2, \ldots, x_n$ are placed at one node (not y) with the global optimal solution. In this case, the optimal objective value can be obtained by fetching $x_1, x_2, \ldots, x_n$ from that node

$$\begin{aligned} L_{\text{opt}} = \sum_{p \in \mathcal{P}} R_{py} \lambda \cdot \mathbf{1}(x_1, x_2, \ldots, x_n \in p) \\ + (R_{x_1y} + R_{x_2y} + \cdots + R_{x_ny}) \cdot \eta. \end{aligned} \tag{19}$$

As the number of requested patterns should be no more than the total number of requested data items, we have

$$\begin{aligned} \sum_{p \in \mathcal{P}} R_{py} \cdot \mathbf{1}(x_1, \ldots, x_n \in p) &\leq \min\{R_{x_1y}, \ldots, R_{x_ny}\} \\ &\leq \frac{R_{x_1y} + \cdots + R_{x_ny}}{n}. \end{aligned} \tag{20}$$

As $n \geq k$, the approximation ratio is bounded by

$$\frac{L}{L_{\text{opt}}} < \frac{(R_{x_1y} + \cdots + R_{x_ny}) \cdot (\eta + \lambda)}{(R_{x_1y} + \cdots + R_{x_ny}) \cdot \eta} < 1 + \frac{\lambda}{\eta}. \tag{21}$$

$v_{x_1y}, v_{x_2y}, \ldots, v_{x_2y} \leq 0$: As shown in Fig. 4b, a full copy of data replicas will be placed at y with the optimal objective value

$$L = L_{\text{opt}} = (W_{x_1} + W_{x_2} + \cdots + W_{x_n}) \cdot \eta. \tag{22}$$

$v_{x_1y}, v_{x_2y}, \ldots > 0$, $x_ny, \ldots \leq 0$: As shown in Fig. 4c, data replica x_n will be placed at node y with the objective value

$$\begin{aligned} L = k_1 \cdot \sum_{p \in \mathcal{P}} R_{py} \lambda \cdot \mathbf{1}(x_1, \ldots, x_n \in p) \\ + (R_{x_1y} + R_{x_2y} + \cdots + W_{x_ny}) \cdot \eta, \end{aligned} \tag{23}$$

where k_1 is the span of involved storage nodes, $k_1 < k$. According to (20), we have

$$L < (R_{x_1y} + R_{x_2y} + \ldots)\lambda + (R_{x_1y} + R_{x_2y} + \cdots + W_{x_ny})\eta. \tag{24}$$

The optimal objective value can be obtained by placing all data replicas at y with

$$L_{\text{opt}} = (W_{x_1} + W_{x_2} + \cdots + W_{x_n}) \cdot \eta. \tag{25}$$

From (17) and (24), the approximation ratio is given by

$$\frac{L}{L_{\text{opt}}} < \frac{(R_{x_1y} + R_{x_2y} + \cdots + W_{x_n})(\eta + \lambda)}{(R_{x_1y} + R_{x_2y} + \cdots + W_{x_n})\eta} = 1 + \frac{\lambda}{\eta}. \tag{26}$$

$v_{x_1y}, v_{x_2y}, \ldots \leq 0, x_ny, \ldots > 0$: As shown in Fig. 4d, the objective value is given by:

$$\begin{aligned} L = k_2 \cdot \sum_{p \in \mathcal{P}} R_{py} \lambda \cdot \mathbf{1}(x_1, \ldots, x_n \in p) \\ + (W_{x_1y} + W_{x_2y} + \cdots + R_{x_ny}) \cdot \eta, \end{aligned} \tag{27}$$

where k_2 is also the span of involved storage nodes. In this case, the independently stored data x_n should be fetched from external storage nodes. We can check all possibilities and determine that Algorithm 1 outputs the optimal solution $L = L_{\text{opt}}$.

For the other two cases $v_{x_1y}, \ldots > 0$, $v_{x_2y}, x_ny, \ldots \leq 0$ and $v_{x_1y}, \ldots \leq 0$, $v_{x_2y}, x_ny, \ldots > 0$, it can be also determined that Algorithm 1 outputs the optimal solution $L = L_{\text{opt}}$. Therefore, the overall worst-case performance bound is given by

$$\frac{L}{L_{\text{opt}}} < 1 + \frac{\lambda}{\eta}. \tag{28}$$

□

5 Online Community Adjustment for Bursty Requests

When the read/write request rates change, it is not necessary to modify the storage system by completely overriding the existing replica placement. An online community adjustment scheme is proposed to handle the bursty requests. Therefore, the service overhead L can be minimized based on the existing replica placement $\{\varepsilon_{xy}\}$ and request routing $\{\delta_{pyj}\}$, $x \in \mathcal{M}, y, j \in \mathcal{N}$.

The Discounting Rate Estimator (DRE) [34] could be applied here to construct the real-time request information $\{R^t_{py}\}$ and $\{W^t_x\}$. For the data read R^t_{py}, a counter is maintained for each pattern p at every node y, which increases with every read operation, and decreases periodically with a ratio factor. Similarly, for the data write W^t_x, a counter is also maintained for each data item x at every node y. The benefits of DRE are as follows: (1) It reacts quickly to the changes of the request patterns; (2) It only requires $O(1)$ space and update time to maintain the prediction for each counter.

The pseudo code of the online community adjustment is listed in Algorithm 2, which monitors all data items from $t = 0$ to T in parallel. Let t' denote the last time that the storage location of data item x is updated at node y. If the variations of read/write request rates are greater than a predefined value, i.e., $\sum_{p \in P_x} \left| R^t_{py} - R^{t'}_{py} \right| + \left| W^t_y - W^{t'}_y \right| > \phi$, the data storage location will be adjusted accordingly for the adaptive community expansion and reduction. The constant ϕ determines how frequently the community is adjusted. Based on the updated replica placement, the request routing will also be updated.

Algorithm 2. Online Community Adjustment

Input: Dataset $\mathcal{M}$, node set $\mathcal{N}$, real-time read/write rate R^t_{py} and W^t_x, master node y_x, existing replica placement ε_{xy} and request routing δ_{pyj}.
Output: Updated placement ε_{xy} and request routing δ_{pyj}.

1: Monitor data item x at node y from $t = 0$ to T, $x \in \mathcal{M}$;
2: **if** $\sum_{p \in P_x} \left| R^t_{py} - R^{t'}_{py} \right| + \left| W^t_y - W^{t'}_y \right| > \phi$ **then**
3: **if** $\varepsilon_{xy} = 1$ && $y \neq y_x$ && $R^t_{xy} < W^t_x$ **then**
4: $\varepsilon_{xy} \leftarrow 0$, $x \notin D_y$;
5: Calculate θ'_{xpy} based on (29), $\forall p \in P_x$;
6: Update routing: $\delta_{pyy_x} \leftarrow 1$ if $\theta'_{xpy} = 1$, $\forall p \in P_x$;
7: **end if**
8: **if** $\varepsilon_{xy} = 0$ **then**
9: Calculate θ_{xpy} based on (14), $\forall p \in P_x$;
10: $\varepsilon_{xy} \leftarrow 1$, $D_y \leftarrow x$, if $v_{xy} \leq 0$;
11: **end if**
12: If replica x is added/removed at t, node y broadcasts the message ε_{xy} to other nodes;
13: **end if**
14: **if** Receive the message $\varepsilon_{xj} = 1$ or $\varepsilon_{xj} = 0, \delta_{pyj} = 1$ from node j, $j \in \mathcal{N}$, $j \neq y$ **then**
15: Update request routing $\{\delta_{pyj}\}$ with the greedy method in Section 4.2, $\forall p \in P_x$;
16: **end if**

Community Reduction (Step 3–7). Initially, the scenario that data replica x is placed at node y is considered, $\varepsilon_{xy} = 1$, $y \neq y_x$. If the read rate decreases with the increased write rate $R^t_{xy} < W^t_x$, replica x is deleted to reduce the total internode traffic without considering the multi-get hole effect. Then, the request routing for x at node y should be updated. Therefore, θ'_{xpy} is introduced to decide whether a new routing destination should be added ($\theta'_{xpy} = 1$) or not ($\theta'_{xpy} = 0$) if replica x is deleted from y

$$\theta'_{xpy} = 1 - \min\{1, \sum_{j \in \mathcal{N}, j \neq y} \delta_{pyj} \cdot \mathbf{1}(\varepsilon_{xy} = 1)\}, \tag{29}$$

where $\delta_{pyj} \cdot \mathbf{1}(\varepsilon_{xy} = 1) = 1$ indicates that the rest routing destinations ($\delta_{pyj} = 1$, $j \neq y$) can still provide data x for R_{py}, $p \in P_x$. If no destination nodes can provide x, node y can fetch x from the master node y_x, $\delta_{pyy_x} \leftarrow 1$.

Community Expansion (Step 8–11). If $\varepsilon_{xy} = 0$, the expansion criterion proposed in Section 4.2 is used to determine whether replica x should be added at node y or not.

Routing Update (Step 12–15). If replica x is added or removed at t, node y broadcasts the message ε_{xy} to all other nodes for routing update. On the other hand, node y may receive the message from other nodes that the replica placement of data item x has been changed. If 1) replica x is added at node j, $\varepsilon_{xj} = 1$, or 2) replica x is removed from the routing destination, $\varepsilon_{xj} = 0$, $\delta_{pyj} = 1$, the routing of all patterns $p \in P_x$ will be updated with the greedy method, $j \in \mathcal{N}$, $j \neq y$. The theoretical analysis of the proposed online community adjustment algorithms is provided as follows.

Property 2. *For a data item x, if the variations of request rates are greater than ϕ, Algorithm 2 will be triggered with the computation complexity of $O(|\mathcal{P}_x| \cdot N)$. If the storage node receives the message that the replica placement of data item x has been changed, Algorithm 2 has the computation complexity of $O(|\mathcal{P}_x| \cdot N^2)$.*

Proof. For data replica x placed at node y, if $R^t_{xy} < W^t_x$, the data item is directly removed. To calculate θ'_{xpy} for routing update, $\forall p \in P_x$, all routing destinations ($\delta_{pyj} = 1$, $j \neq y$) should be considered with $|\mathcal{P}_x| \cdot N$ iterations at most. Then, if x is not placed at node y, the community expansion also needs $|\mathcal{P}_x| \cdot N$ iterations to decide whether x should be placed at y, just as discussed in Property 1. The computation complexity of Algorithm 2 for a data item is $O(|\mathcal{P}_x| \cdot N)$ when the variations of request rates are greater than ϕ.

Then, if node y receives the message that the storage location of x has been changed, the routing for all $|\mathcal{P}_x|$ request patterns is updated with the greedy method. The computation complexity of Algorithm 2 is $O(|\mathcal{P}_x| \cdot N^2)$. □

Theorem 3. *At time t, Algorithm 2 incurs no more than $1 + \frac{\lambda}{\eta}$ times of the optimal objective value L^t_{opt}.*

Proof. Initially, the community reduction process ensures that $R^t_{xy} \geq W^t_x$ holds for all remaining data replicas at each node. The minimum number of data replicas can be achieved for data item x to reduce the total service overhead L^t at time t. Then, by considering the multi-get

hole effect, the community expansion process gradually adds more data replicas at each node. According to Theorem 2, the worst-case performance bound of the incremental community expansion is $1+\frac{\lambda}{\eta}$. By using the same expansion criterion, the worst-case performance bound of the online community adjustment algorithms is also given by

$$\frac{L^t}{L^t_{\text{opt}}} < 1+\frac{\lambda}{\eta}. \tag{30}$$

□

Guided by the offline community discovery scheme, the proposed online community adjustment scheme is also a distributed solution. All storage nodes only need to monitor the information of data read/write request rates and update the replica placement in parallel. Compared with the centralized solution which needs to collect the global real-time request information for placement decision making, the communication overhead of our scheme is limited with a higher scalability.

6 Performance Evaluation

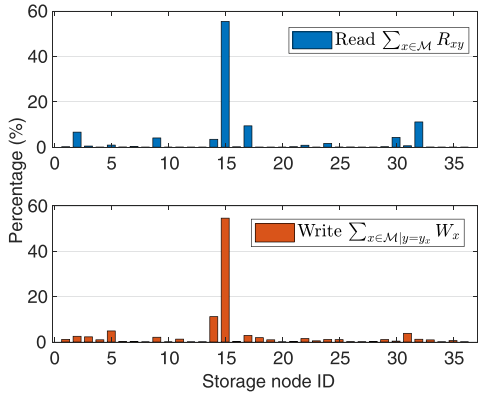
In this section, extensive evaluations are performed to evaluate the performance of the proposed distributed community discovery (DCD) and online community adjustment (OCA) algorithm. Two real-world traces, i.e., MSR Cambridge Traces [35] and Facebook Friendships Dataset [36], are synthesized together to extract the real-time request information for associated data items. The evaluations are conducted in Python for a parallel implementation of the geo-distributed storage system.

6.1 Trace Description and Experiment Settings

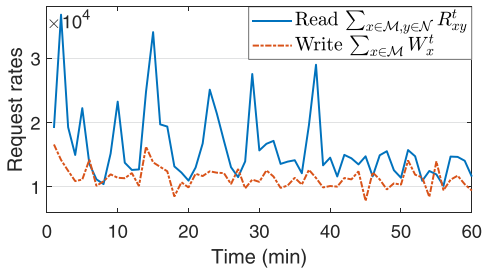
MSR Cambridge Traces [35]. These are the I/O traces gathered from servers located at Microsoft Research Cambridge where data read/write requests are captured from 36 storage volumes for a week. Based on the traces, it can be assumed that there are $N=36$ geo-distributed storage nodes in the experiments. Fig. 5a illustrates the arrival rates of the requests for a period of $T=1$ hour at each node. The distribution is biased among storage nodes due to real applications. The proposed DCD algorithm can find a replica placement solution based on the average read/write rates for this period. For each request, the hostname, request type (read/write), and timestamp are given. Fig. 5b illustrates the time-varying feature of data request rates. Based on DCD, the OCA algorithm is further proposed to handle the bursty requests during the period T.

However, for the reason of confidentiality, most publicly available traces, including the utilized MSR Cambridge Traces, do not specify the detailed data items for each read/write request. Therefore, Facebook Friendships Dataset is introduced. These two data traces are synthesized together to extract the detailed data request information.

Facebook Friendships Dataset [36]. Similar to the previous study [5], the request pattern information is extracted from the OSN dataset. The friend relationships can be discovered through dataset analysis. Let us consider that a news update operation from a user would incur the data fetching of all



(a) Read/write request arrival rates at each node



(b) Time-varying read/write request rates from all nodes

Fig. 5. Read/write request rates based on MSR Cambridge Traces [35].

his friends. The friend relationships form the request pattern information of each considered user. Each user is treated as a data item. Then, there are $M=63,731$ data items and $|\mathcal{P}|=45,092$ read request patterns in total. Each request pattern contains 13.92 items on average. It can be assumed that each data item is randomly and uniformly assigned to one of the 36 nodes as the original storage location. The read rates of request patterns and the write rates of data items at node y follow a Zipf distribution with $R_y=\sum_{p\in\mathcal{P}} R_{py}$ and $W_y=\sum_{x\in\mathcal{M}} W_x$ as shown in Fig. 5a, just similar to [5], [8], [17]. The tail index of the Zipf distribution is set to 1.25. The ratio of read/write frequency of all data items $\frac{\sum_{x\in M}\sum_{y\in N} R_{xy}}{\sum_{x\in M} W_x}$ is 19.2. Please note that the proposed data replica placement schemes can be evaluated under various system configurations. Experiment settings, e.g., various workload characterizations, data request distributions, and autoregressive models, can be configured according to different user choices.

Furthermore, the default values of overhead λ and weight η are set to 1.0 and 1.0, respectively. The constant ϕ is set to 10 for OCA.

Performance Baselines. In the experiments, three other data replica placement schemes are introduced for a fair performance comparison. The first is Random—randomly places data replicas to storage nodes to optimize the load-balancing, which has been widely adopted in the distributed storage systems today, such as HDFS [28] and Cassandra [29]. The second is ADP [5]—utilizes hypergraph partitioning to iteratively optimize the storage locations of data replicas. ADP is a centralized scheme to reduce the access overhead of associated data items and the inter-node read traffic. The third is SDP [5]—constructs a hypergraph sparsifier of data

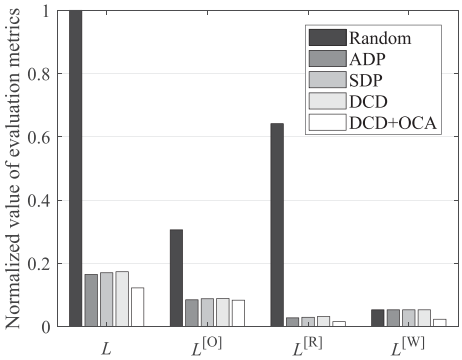


Fig. 6. Performance comparison with other schemes.

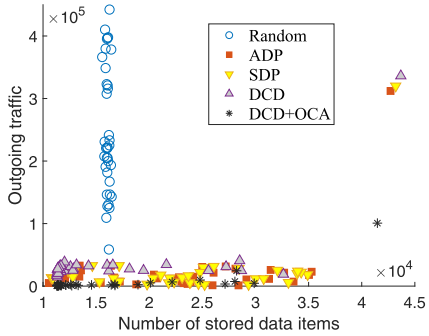


Fig. 7. Number of stored data items and outgoing traffic of nodes in a scatter plot.

traffic to lower the computation complexity of ADP. However, the synchronization traffic for data write is ignored in ADP and SDP.[1]

6.2 Evaluation Results

In the beginning, the performances of four offline schemes, i.e., Random, ADP, SDP, and DCD, are compared in terms of the obtained objective values L based on their replica placement results. It is worth noting that all the objective values shown in the following figures are normalized against the worst performance obtained by Random under the default settings. Considering the associated relationship of data in the request patterns, ADP, SDP, and DCD generate less inter-node read traffic $L^{[R]}$ and access overhead $L^{[O]}$ in comparison with Random. As shown in Fig. 6, compared with Random, the overall objective value with ADP, SDP and DCD can be reduced by 83.52, 83.03 and 82.67 percent, respectively.

Furthermore, ADP and SDP are centralized schemes, which use hypergraph partitioning and sparsification to iteratively achieve the replica placement solution, respectively. Therefore, ADP and SDP achieve less $L^{[R]} + L^{[O]}$ than that achieved by the distributed scheme DCD. Note that the data synchronization traffic $L^{[W]}$ is the same with ADP, SDP, and DCD in Fig. 6. Hence, the performance gap between ADP, SDP, and DCD is limited. However, the computation overheads of ADP and SDP are higher than that of DCD, just as shown in Section 6.3.

Then, the performance of the proposed online scheme OCA is compared. Based on the replica placement results obtained by DCD, OCA handles the bursty requests in Fig. 5b. The replicas of data items are adaptively added or removed from storage nodes according to the current data read/write request rates. Compared with the proposed offline solution DCD, the overall objective value can be further reduced by 29.48 percent with DCD + OCA.

The load balancing performance is also compared. Fig. 7 illustrates the number of stored data items and outgoing traffic of each storage node with different solutions. Due to the stochastic nature of Random, the stored data items are uniformly distributed among nodes. However, due to the biased distribution of user requests, the outgoing traffic is unbalanced. Note that ADP, SDP, and DCD intend to place more replicas on nodes with more read requests to reduce the inter-node read traffic. The stored data items are unevenly distributed but with more balanced outgoing traffic. Furthermore, OCA tracks the dynamics of data request rates. More data replicas will be added to further reduce the read traffic when the data read requests increase. On the contrary, unnecessary data replicas will be removed to reduce the synchronization traffic when the data write dominates. Therefore, both the stored data items and the outgoing traffic become more balanced with DCD + OCA.

6.3 Network Scalability

The scalability of the placement scheme is determined by its computation complexity. Four offline solutions, i.e., Random, ADP, SDP, and DCD, are included for a fair performance comparison in Figs. 8 and 9. The average running time of four offline schemes determines the efficiency of deploying a replica placement solution. Please note that OCA monitors the dynamics of request rates for each data item during the service period T in a real-time manner. Therefore, OCA is not included in the performance comparison. The hardware in the experiments features an Intel(R) Core(TM) i7-7700 HQ processor and 16 GB memory.

Compared with any data placement solutions, Random achieves the minimum computation complexity by using simple heuristics. Under the default settings, Random only lasts 2.18 s on average. Due to the high complexity of hypergraph partitioning over large-scale networks, ADP needs a

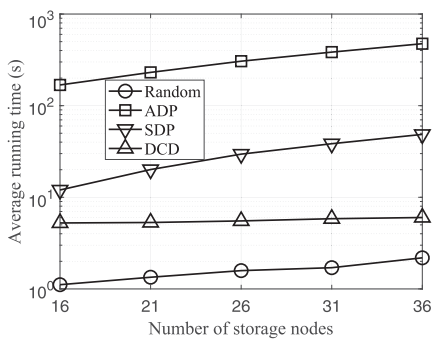


Fig. 8. Network scalability of the offline schemes: Average running time with the number of storage nodes.

1. In the design of Random, ADP, and SDP, the replica number of data items is a predefined value. Therefore, for a fair performance comparison, the replica number of Random, ADP, and SDP is set to the same value achieved in DCD. Under this setting, the data synchronization traffic is the same with Random, ADP, SDP, and DCD.

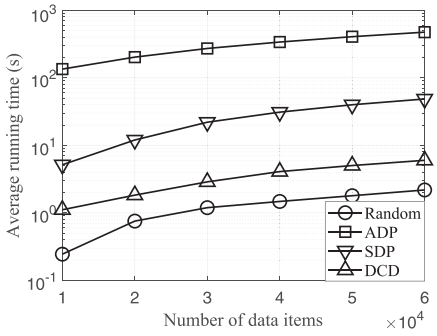


Fig. 9. Network scalability of the offline schemes: Average running time with the number of data items.

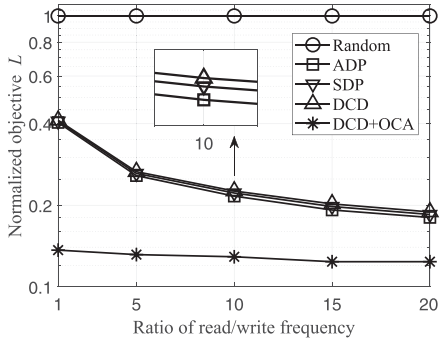


Fig. 10. Impact of the read/write frequency.

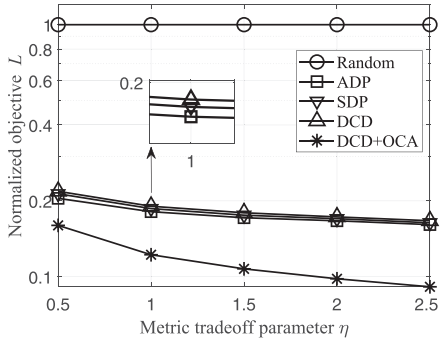


Fig. 11. Impact of tradeoff parameter η.

much longer time of 474.62 s to converge.[2] Furthermore, SDP sparsifies the hypergraph through probabilistic sampling, reducing the computation complexity of hypergraph partitioning with a running time of 48.58 s. Unlike the centralized schemes above, with the parallel community expansion, the running time of DCD is determined by the longest running time of the individual data-node community. Compared with ADP, DCD incurs a similar overhead L with a much lower running time of 6.01 s.

Then, the scalability is evaluated with different network scales where the number of storage nodes N varies from 16 to 36, with the fixed number of data items $M = 63,731$ and read request patterns $|\mathcal{P}| = 45,092$, just as mentioned in Section 6.1. This can be realized by using part of the MSR Cambridge Traces. As shown in Fig. 8, the average running time increases dramatically from 154.34 to 474.62 s with the centralized ADP scheme. Although with reduced computation complexity, the average running time of SDP also increases rapidly from 12.02 to 48.58 s. In contrast, with the distributed DCD scheme, the average running time increases slightly from 5.24 to 6.01 s. This means the proposed DCD scheme is more scalable for the large-scale deployments of storage nodes.

Fixing the number of storage nodes at 36, the running time performance is evaluated with the number of data items varying from 10,000 to 60,000. Similarly, this can be realized by extracting part of data items from the Facebook Friendships Dataset. As shown in Fig. 9, the average running time increases from 134.64 to 472.28 s with ADP and from 5.16 to 48.17 s with SDP. Furthermore, according to our design, the computation complexity of DCD is linear to the amount of data items. Therefore, the average running time with DCD also increases linearly from 1.11 to 6.0 s. This demonstrates the scalability and efficiency of our scheme DCD to handle large-scale datasets.

6.4 Impact of Other Factors

To fully evaluate the performance of the proposed DCD and OCA schemes, several other factors are considered with the fixed setting $N = 36$, $M = 63,731$, and $|\mathcal{P}| = 45,092$.

Read/Write Frequency. The data access type varies for different storage systems. For example, the system could be both read-intensive and write-intensive for OSN. On the other hand, for content delivery networks, the system is read-intensive with rare data write. Therefore, the write frequencies of data items W_x are magnified or reduced in equal proportion to emulate the scenario with various data access types. When the ratio of read/write frequency is 1.0, the average number of created replicas for each data item is 9.07 with DCD, ADP and Random. Meanwhile, DCD + OCA generates 9.26 replicas on average. Then, when the ratio of read/write frequency increases to 20.0, more data replicas (11.48 with DCD, ADP, SDP, and Random, and 11.78 with DCD + OCA) will be created to reduce the inter-node traffic and the access overhead to associated data. With more data replicas, ADP, SDP, DCD, and OCD schemes have more space to further reduce the service overhead L. As shown in Fig. 10, the performance gaps between Random and the other three schemes become higher and higher. For example, the performance gain of DCD over Random increases from 58.81 to 82.84 percent with the increased read/write ratio.

Weight η. The tradeoff parameter η in (9) can balance the importance of the data association and inter-node traffic. With the increase of η from 0.5 to 3, the weight of the inter-node traffic goes up in the objective function. According to the performance bound in Eq. (28), with the increase of η, the output of DCD is getting closer and closer to the optimal solution.[3] As shown in Fig. 11, the performance gain of DCD over Random increases from 78.18 to 83.33 percent with the increased weight η.

Pre-Defined Value ϕ. The pre-defined value ϕ determines how frequently the community is adjusted, which greatly

2. The running time of ADP is not the same as that in [5] due to the difference of used hardware and parameter settings in the experiments.

3. Note that if $\eta \to \infty$, then the access overhead to associated data is not considered, and DCD can achieve the optimal objective value.

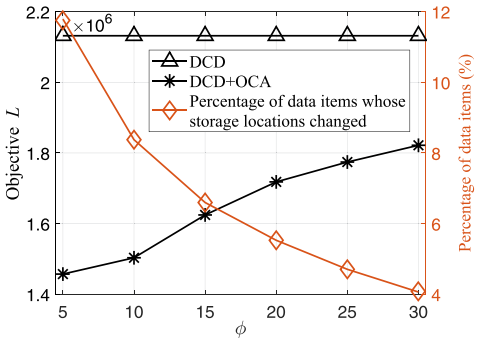


Fig. 12. Impact of ϕ on the online scheme OCA.

influences the performance of the proposed online scheme OCA. As the read/write rates of data items follow a Zipf distribution, a small portion of data items occupies the majority of data requests. This also means that during the service period, the variations of request rates are not significant for the remaining large number of data items. When ϕ increases from 5 to 30, the percentage of data items that have changed their storage locations during the service period is decreased from 11.79 to 4.06 percent. The data-node communities are less frequently adjusted with less computation overhead. As shown in Fig. 12, compared with the offline scheme DCD, the performance gain with DCD + OCA is decreased from 31.65 to 14.54 percent along with the increased value ϕ.

Furthermore, Fig. 12 demonstrates that the proposed online scheme OCA can reduce the computation overhead when faced with bursty requests. It is not necessary to completely overriding the existing replica placement with the offline scheme DCD. Compared with DCD, only part of the data items will change their storage locations with DCD + OCA.

7 Conclusion and Future Work

Observing the increasing scale of data items and time-varying data requests in geo-distributed storage systems, we proposed scalable and adaptive data replica placement schemes based on the overlapping community discovery approach to improve the efficiency of making placement decisions. With an overall consideration of the inter-node traffic and system overhead of accessing associated data, data-node communities can evolve to decide whether each data replica should be placed at each node in a parallel way. This distributed implementation along with the linear computation complexity over the number of data items ensures the scalability of our design. The online scheme was further proposed to adaptively handle the bursty requests. The worst-case performance bound was also theoretically analyzed. Evaluation driven by real-world datasets showed that compared with the centralized scheme ADP, the proposed scheme DCD incurs similar data access overhead, while greatly reduces the running time. Guided by the offline DCD, the data access overhead can be further reduced by about 30 percent with the online OCA.

In future work, more performance metrics, e.g., data access latencies, cost of storage, and load balance among storage nodes, will be considered in the data replica placement. These metrics may also influence the performance of the storage system. Furthermore, a prototype of the geo-distributed cloud storage system based on Amazon EC2 clusters will be built for a series of real-world experiments to validate the performance of the proposed data replica placement schemes.

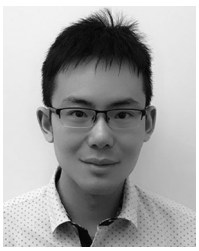
Acknowledgments

The authors would like to acknowledge that this work was partially supported by the National Natural Science Foundation of China (Grant Nos. 61672537, 61672539 and 61873353), China Postdoctoral Science Foundation, and in part by NSERC, CFI, and BCKDF.

References

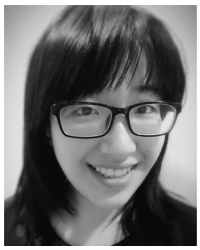
[1] Data Growth, Business Opportunities, and the IT Imperatives, 2014. [Online]: https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm
[2] Y. Mansouri, A. N. Toosi, and R. Buyya, "Data storage management in cloud environments: Taxonomy, survey, and future directions," *ACM Comput. Surv.*, vol. 50, no. 6, 2017, Art. no. 91.
[3] G. Liu, H. Shen, and H. Chandler, "Selective data replication for online social networks with distributed data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 8, pp. 2377–2393, Aug. 2016.
[4] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2010, Art. no. 2.
[5] B. Yu and J. Pan, "A framework of hypergraph-based data placement among geo-distributed datacenters," *IEEE Trans. Services Comput.*, to be published, doi: 10.1109/TSC.2017.2712773.
[6] Q. Pu *et al.*, "Low latency geo-distributed data analytics," in *Proc. ACM Conf. Special Interest Group Data Commun.*, 2015, pp. 421–434.
[7] B. Yu and J. Pan, "Sketch-based data placement among geo-distributed datacenters for cloud storages," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.
[8] X. Ren, P. London, J. Ziani, and A. Wierman, "Datum: Managing data purchasing and data placement in a geo-distributed data market," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 893–905, Apr. 2018.
[9] L. Jiao, J. Li, W. Du, and X. Fu, "Multi-objective data placement for multi-cloud socially aware services," in *Proc. IEEE INFOCOM*, 2014, pp. 28–36.
[10] A. Charapko, A. Ailijiang, and M. Demirbas, "Adapting to access locality via live data migration in globally distributed datastores," in *Proc. IEEE Int. Conf. Big Data*, 2018, pp. 3321–3330.
[11] D. A. Tran, K. Nguyen, and C. Pham, "S-CLONE: Socially-aware data replication for social networks," *Comput. Netw.*, vol. 56, no. 7, pp. 2001–2013, 2012.
[12] S. Traverso, K. Huguenin, I. Trestian, V. Erramilli, N. Laoutaris, and K. Papagiannaki, "TailGate: Handling long-tail content with a little help from friends," in *Proc. 21st Int. Conf. World Wide Web*, 2012, pp. 151–160.
[13] S. Raindel and Y. Birk, "Replicate and bundle (RnB)–A mechanism for relieving bottlenecks in data centers," in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process.*, 2013, pp. 601–610.
[14] R. Nishtala *et al.*, "Scaling memcache at Facebook," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2013, pp. 385–398.
[15] A. Atrey, G. V. Seghbroeck, H. Mora, F. D. Turcka, and B. Volckaert, "SpeCH: A scalable framework for data placement of data-intensive services in geo-distributed clouds," *J. Netw. Comput. Appl.*, vol. 142, pp. 1–14, 2019.
[16] V. Zakhary, F. Nawab, D. Agrawal, and A. E. Abbadi, "Global-scale placement of transactional data stores," in *Proc. 21st Int. Conf. Extending Database Technol.*, 2018, pp. 385–396.
[17] K. Liu, J. Wang, Z. Liao, B. Yu, and J. Pan, "Learning-based adaptive data placement for low latency in data center networks," in *Proc. IEEE 43rd Conf. Local Comput. Netw.*, 2018, pp. 142–149.
[18] K. Liu *et al.*, "A learning-based data placement framework for low latency in data center networks," *IEEE Trans. Cloud Comput.*, to be published, doi: 10.1109/TCC.2019.2940953.
[19] M. Annamalai *et al.*, "Sharding the shards: Managing datastore locality at scale with Akkio," in *Proc. USENIX Conf. Operating Syst. Des. Implementation*, 2018, pp. 445–460.

[20] S. Sobolevsky and R. Campari, "General optimization technique for high-quality community detection in complex networks," *Phys. Rev. E*, vol. 90, no. 1, 2014, Art. no. 012811.
[21] V. D. Blondel, J. L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Statist. Mech.-Theory Experiment*, vol. 2008, no. 10, pp. 1–12, 2008.
[22] N. P. Nguyen, T. N. Dinh, Y. Xuan, and M. T. Thai, "Adaptive algorithms for detecting community structure in dynamic social networks," in *Proc. IEEE INFOCOM*, 2011, pp. 2282–2290.
[23] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, pp. 814–818, 2005.
[24] H. Long, "Overlapping community detection with least replicas in complex networks," *Inf. Sci.*, vol. 453, pp. 216–226, 2018.
[25] H. Chen, H. Jin, and S. Wu, "Minimizing inter-server communications by exploiting self-similarity in online social networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 4, pp. 1116–1130, Apr. 2016.
[26] J. M. Pujol *et al.*, "The little engine(s) that could: Scaling online social networks," in *Proc. ACM SIGCOMM Conf.*, 2011, pp. 375–386.
[27] K. Hu and G. Zeng, "Placing big graph into cloud for parallel processing with a two-phase community-aware approach," *Future Gener. Comput. Syst.*, vol. 101, pp. 1187–1200, 2019.
[28] HDFS Architecture Guide, 2019. [Online]: https://hadoop.apache.org/
[29] Cassandra, 2019. [Online]: http://cassandra.apache.org/
[30] J. S. Hunter, "The exponentially weighted moving average," *J. Quality Technol.*, vol. 18, no. 4, pp. 203–210, 1986.
[31] J. Zhou and J. Fan, "JPR: Exploring joint partitioning and replication for traffic minimization in online social networks," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 1147–1156.
[32] M. R. Garey and D. S. Johnson, *Computer and Intractability: A Guide to The Theory of NP-Completeness*. Amsterdam, The Netherlands: Elsevier, 1979.
[33] U. Feige, "A threshold of ln n for approximating set cover," *J. ACM*, vol. 45, no. 4, pp. 634–652, 1998.
[34] M. Alizadeh *et al.*, "CONGA: Distributed congestion-aware load balancing for datacenters," *ACM SIGCOMM Comput. Commun. Rev.*. vol. 44, pp. 503–514, 2014.
[35] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Trans. Storage*, vol. 4, no. 3, pp. 1–23, 2008.
[36] J. Kunegis, "KONECT: The Koblenz network collection," in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 1343–1350.



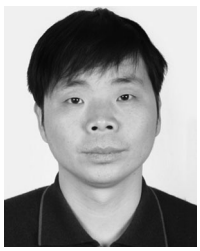
Kaiyang Liu (Member, IEEE) received the PhD degree from the School of Information Science and Engineering, Central South University, Changsha, China, in 2019. From 2016 to 2018, he was a research assistant with the University of Victoria, Victoria, BC, Canada. His research interests include networked systems, distributed systems, and cloud/edge computing, with a special focus on the analysis and optimization of the data-intensive services. He is one of the three IEEE LCN 2018 Best Paper Award candidates.

Jun Peng (Member, IEEE) received the BS degree from Xiangtan University, Xiangtan, China, in 1987, the MSc degree from the National University of Defense Technology, Changsha, China, in 1990, and the PhD degree from Central South University, Changsha, China, in 2005. In April 1990, she joined the Central South University. From 2006 to 2007, she was with the School of Electrical and Computer Science, University of Central Florida, as a visiting scholar. She is currently a professor with the School of Computer Science and Engineering, Central South of University, China. Her research interests include cooperative control, cloud computing, and wireless communications.

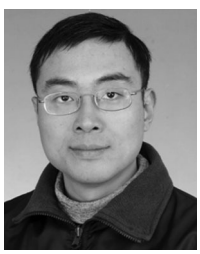


Jingrong Wang (Student Member, IEEE) received the bachelor's degree from the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, China, in 2017, and the MSc degree in computer science from the University of Victoria, Victoria, BC, Canada, in 2019. She is currently working toward the PhD degree at the University of Toronto, Toronto, ON, Canada. Her research interests include wireless communications, mobile edge computing, and machine learning.

Weirong Liu (Member, IEEE) received the BE degree in computer software engineering and the ME degree in computer application technology from the Central South University, Changsha, China, in 1998 and 2003, respectively, and the PhD degree in control theory and control engineering from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2007. Since 2008, he has been a faculty member with the School of Information Science and Engineering, Central South University, where he is currently a professor. His research interests include cooperative control, energy storage management, reinforcement learning, neural networks, wireless sensor networks, network protocol, and microgrids.



Zhiwu Huang (Member, IEEE) received the BS degree in industrial automation from Xiangtan University, Xiangtan, China, in 1987, the MS degree in industrial automation from the Department of Automatic Control, University of Science and Technology Beijing, Beijing, China, in 1989, and the PhD degree in control theory and control engineering from Central South University, Changsha, China, in 2006. In 1994, he joined the Central South University. From 2008 to 2009, he was with the School of Computer Science and Electronic Engineering, University of Essex, United Kingdom, as a visiting scholar. He is currently a professor with the School of Automation, Central South University, China. His research interests include fault diagnostic technique and cooperative control.



Jianping Pan (Senior Member, IEEE) received the bachelor's and PhD degrees in computer science from Southeast University, Nanjing, Jiangsu, China. He is currently a professor of computer science with the University of Victoria, Victoria, British Columbia, Canada. He did his postdoctoral research with the University of Waterloo, Waterloo, Ontario, Canada. He also worked with Fujitsu Labs and NTT Labs. His area of specialization is computer networks and distributed systems, and his research interests include protocols for advanced networking, performance analysis of networked systems, and applied network security. He received the IEICE Best Paper Award in 2009, the Telecommunications Advancement Foundation's Telesys Award in 2010, the WCSP 2011 Best Paper Award, the IEEE Globecom 2011 Best Paper Award, the JSPS Invitation Fellowship in 2012, the IEEE ICC 2013 Best Paper Award, and the NSERC DAS Award in 2016, and is a co-author of one of the three IEEE LCN 2018 Best Paper Award candidates, and has been serving on the technical program committees of major computer communications and networking conferences including IEEE INFOCOM, ICC, Globecom, WCNC, and CCNC. He was the ad hoc and Sensor Networking Symposium co-chair of IEEE Globecom 2012 and an associate editor of the *IEEE Transactions on Vehicular Technology*.