This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2022.3155357, IEEE Transactions on Network Science and Engineering

Pushing Collaborative Data Deduplication to the Network Edge: An Optimization Framework and System Design

Shijing Li, Tian Lan, Bharath Balasubramanian, Hee Won Lee, Moo-Ryong Ra, Rajesh Panta

Abstract—Edge computing has become a new computing paradigm with explosive growth in recent years. We consider the problem of pushing data deduplication to the network edge and propose a new framework for distributed edge-facilitated deduplication (EF-dedup). Deduplication at the network edge allows us to exploit the high degree of geographic- and temporal-correlation in edge data to achieve space efficiency. By leveraging distributed computing power available on the edge in a collaborative fashion, the edge nodes can effectively suppress duplicated edge data, consuming considerably less space and WAN bandwidth. To this end, we partition the edge nodes into disjoint collaborative clusters, maintain a deduplication index structure across them using a distributed key-value store and perform deduplication within those clusters. However, this partitioning problem is very challenging and requires the optimization of a novel tradeoff: edge nodes with highly correlated data may not always be within the same edge cloud, with non-trivial network cost among them. We formulate a joint storage and network optimization problem with different design objectives, such as arbitrary partitioning and balanced partitioning of edge nodes. The problem is shown to be NP-Hard in general. Then, an optimization framework with efficient algorithms is developed and is proven to achieve a closed-form competitive ratio. Our experiments, performed on edge nodes in a corporate lab and a central cloud at AWS, demonstrate that EF-dedup achieves 67.4∼133.7% better deduplication throughput than sole cloud-based techniques and achieves 20.0-62.6% lesser aggregate cost in terms of the network-storage trade-off as compared to approaches that solely favor one over the other.

Index Terms—Distributed storage, Data deduplication, Edge computing.

1 Introduction

Due to the emergence of IoT and edge computing, large volumes of data - exhibiting high degree of geographic- and temporal-correlations. - would be continuously generated by smart mobile devices, connected cars, sensors, etc [1], [2], [3]. Recent market analysis results show that the stored in the edge-facing IoT devices is expected to reach 5.9 ZB by 2021 [4] and crucially, 43% of such data will be processed in edge clouds [5], while the remaining data would be sent to the central cloud. Given such highly correlated data generated at the network edge, it presents a unique opportunity to push data deduplication to the network edge, not only to improve the space efficiency of edge storage, but also to mitigate the WAN bandwidth drain for sending the massive amounts of data to the cloud.

Data deduplication, which is a well studied [6], commercially applied technique [7], [8], is the process of splitting files into smaller chunks and storing only unique chunks. It has been shown to significantly reduce storage space not only for traditional cloud workloads like VM images, but also for IoT data such as traffic video image sequences and car multimedia system images (by up to 76-84%) [9][10][11]. In this paper, we develop an optimization framework and system design for pushing deduplication - typically employed as a cloud-centric solution in modern datacenters- to the network edge. This approach has several advantages. First, deduplicating close to data sources eliminates duplicates at an earlier stage, thereby significantly reducing both the wide-are-network (WAN) bandwidth needed to transmit data to the cloud and the burden for expensive cloud uplink provisioning.

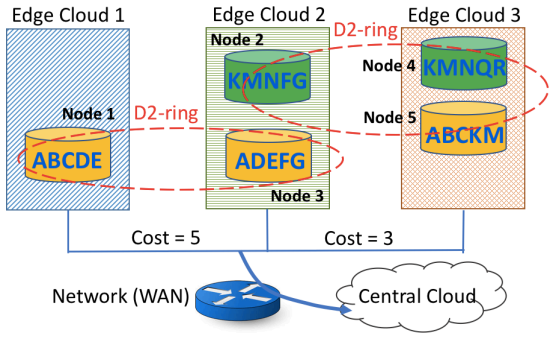


Fig. 1. An example of collaborative, edge-centric deduplication and the associated network-storage trade-off.

Shijing Li and Tian Lan are with the Department of Electrical and Computer Engineering, George Washington University.
Bharath Balasubramanian, Hee Won Lee, Moo-Ryong Ra, and Rajesh Panta were with the AT&T Labs Research.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2022.3155357, IEEE Transactions on Network Science and Engineering

Second, by leveraging the network and computing power on "everything" at the edge we can achieve much higher deduplication throughput[1] than approaches in which we maintain the deduplication index structure (hashes of commonly appearing chunks) solely at the central cloud and perform remote lookups which could incur hundreds of milliseconds. In our experiments, cloud assisted approaches have 56% less throughput than our approach due to WAN latency. Third, data flows generated by IoT devices are often geographically correlated, e.g., sensors operating in the same environment or cameras located in close vicinity and therefore edge deduplication provides the promise of significant space efficiency.

Typical implementations of deduplication, however, require operators to allocate dedicated hardware in the form of appliances with a carefully engineered network and non-trivial system resources (e.g., cpu, memory). Clearly, for the resource-constrained edge nodes (e.g., a half rack deployed in a central office of a city), this is not possible.

Main problem and challenges. The goal of this paper is to develop a framework for minimizing the network cost due to distributed index structure lookup, while maintaining a competitive deduplication ratio (or equivalently minimum storage space cost) despite resource constraints of edge nodes, through the proposed EF-dedup solution and the related optimization of edge node partitioning. However, the problem is shown to be NP-hard. Despite some recent partitioning heuristics like [12], there is no optimization framework to support various design objectives, such as balanced and unbalanced edge node partitioning, or to enable a quantitative analysis of the resulting competitive ratio.

This is a hard problem for three major reasons. First, an intuitive approach to perform collaborative deduplication is to partition the edge nodes into smaller clusters and perform deduplication within those clusters. However, this leads to a non-trivial trade-off between network cost and deduplication ratio (we refer to it as storage efficiency). Consider the five edge nodes of Fig. 1 that are connected by two links with different network costs (based on latency), with data flows comprised of sequences of data chunks that possess different levels of similarity. Clearly, partitioning nodes $\{1, 3, 5\}$ and $\{2, 4\}$ maximizes the deduplication ratio with a total of 16 unique chunks from the two clusters. However, it causes high network cost, in particular, between nodes 1 and 5. On the other hand, deduplicating each edge cloud separately achieves minimum network cost, yet it is not storage efficient with 21 unique chunks. An optimal partitioning (Cluster 1 and 2 in Fig. 1) of edge nodes must account for both network cost and data similarity.

Second, to partition optimally we need to construct efficient models to estimate and track the time-varying similarity across different data sources. Naive approaches that exhaustively search across all the sources will be very time consuming as well as computationally expensive, especially for the edge environment. In this paper, we propose an estimation model to predict the deduplication ratio among files by using periodic samples from the data sets to form characteristic vectors that best represent the statistics of the input data flows. By regularly adjusting the estimation of characteristic vectors across time with new samples, we can restrict the estimation error to less than 4%. Note that this estimation is an offline process that takes less than 4 minutes. Once we have an accurate characteristic vector, we can perform efficient online edge deduplication.

Finally, we need to identify the design techniques and data-structures that will enable distributed edge deduplication. For example, the edge nodes within a partition may have scarce resources and the networks connecting them may be unreliable, especially when they go across edge clouds, as in the first cluster (i.e., Cluster/D2-ring 1) in Fig. 1. Therefore, existing solutions such as a shared network file system across such nodes maybe impractical.

EF-dedup Solution. o address the first challenge, we formulate an optimization problem to partition the edge nodes into collaborative D2-rings, considering: (i) the data similarities across the nodes (i.e., deduplication ratio will improve as more data flows belonging to the same D2-ring are similar), and (ii) the network cost in performing deduplication across edge nodes (i.e., the deduplication throughput will decrease when performing deduplication across edge nodes that are distant from each other). To this end, we capture the data similarity across different edge nodes through a novel, hierarchical data model. In particular, we first construct a set of independent chunk pools each of which can represent commonly occurring data sources, e.g., chunk pools typical of windows, Linux, and word dictionaries. Given these chunk pools, the data flow generated by each edge node can be statistically constructed by randomly drawing data from the chunk pools, according to a pre-determined (empirical) probability distribution function, that can be estimated by sampling the node data. The proposed hierarchical data model is validated using real-world datasets with an average error less than 4% (Sec. 3.1).

Based on this model, we formulate the joint space and network optimization for edge-centric deduplication, which we prove is NP-hard. Then, we develop an optimization framework with efficient algorithms - going beyond arbitrary partitioning heuristics in [12] - for balanced partitioning with better load-sharing among edge nodes. We also prove that the proposed algorithms are optimal when the number of chunks pools is equal to two, and in general, achieves an approximate ratio of $1 + \left(\frac{S}{U^*} - 1\right)\left(1 - \frac{1}{Z}\right)$, where S is the size of all chunk pools, U^* the optimal storage space, and Z the size of D2-rings. Our models and analysis pave the way for a systematic framework for collaborative edge-centric deduplication.

1. Here, the deduplication throughput, as experienced by the clients uploading data, is the amount of input data deduplicated within a certain timeframe.

Implementation and Evaluation. While our algorithms optimize edge-node partitions with provable performance guarantees, another major challenge lies in designing a system to perform distributed deduplication within each resulting partition (i.e., D2-ring), which could often contain nodes across different edge clouds. In EF-dedup we address this challenge through the following intuition: given the resource constraints of individual edge node in each D2-ring and the need for parallel data flow processing, why not maintain the index structure in a distributed key value store (such as Cassandra [13]) spread across the edge nodes in each D2-ring? The distributed storage system provides a way to harness resources (including cpu, memory and storage space) that already exist on edge nodes, in an effective fashion with enormous flexibility and scalability. This empowers EF-dedup with the ability to leverage various resource from nodes at the edge of the network.

We realize EF-dedup by modifying an existing deduplication tool, duperemove [14] to use Cassandra for its index structure. Our experiments on two real-world clouds, one in a corporate lab environment and the other in AWS, with real-world datasets shows that EF-dedup achieves 67.4$\sim$133.7% better deduplication throughput than sole cloud-based techniques and achieves 20.0-62.6% lesser aggregate cost in terms of the network-storage trade-off as compared to approaches that solely favor one over the other.

Key Contributions. In summary, we make the following contributions:

- We motivate the need to push deduplication to the network edge, identify a novel tradeoff in the problem space and formulate an optimization problem to partition edge nodes considering both storage efficiency and network cost (Sec. 2). The formulation is enabled by a new hierarchical model using chunk pools and character vectors for correlated data flows.
- We prove that the formulated problem is NP-Hard and provide efficient algorithms to the problem with bounded approximate performance ratio (Sec. 3.2 and Sec. 3.3).
- We present an end-to-end system design to realize EF-dedup wherein we maintain the index structure of each deduplication D2-ring in a distributed storage system deployed across multiple edge clouds. We implemented EF-dedup by modifying duperemove [14] to use Cassandra for its index structure (Sec. 4).
- We validate our system model and algorithms through extensive simulations and experiments performed on two real clouds using real-world datasets (Sec. 5). Significant improvement on system throughput is achieved yet with high deduplication ratio.

2 System Model and Problem Formulation

We consider a set of N distributed edge nodes, e.g., VMs in cloudlets/fog/edge clouds, denoted by $\mathcal{N} = \{1, 2, \ldots, N\}$. The edge nodes generate data flows, such as VM/system backup, smartphone images and sensing data, which need to be stored in the central cloud. By deduplicating the data at these nodes, the amount of data that must be transferred to the cloud can be greatly reduced, given that the same data chunks may occur frequently in spatially/temporally correlated data flows.

We propose a novel approach to enable distributed deduplication where we partition the edge nodes into disjoint clusters (called D2-rings), that may traverse different edge clouds, based on their network conditions and data correlation. Each D2-ring independently performs deduplication, where unique chunks generated by the nodes in the D2-ring are identified and transferred to the central cloud for data storage. Each D2-ring maintains the deduplication index structure containing hashes of chunks that have been sent to the central cloud. As the process continues, hash values of the unique chunks are stored locally and distributed across all the edge nodes associated with the ring, so that any incoming chunks are compared to the hash values to determine if a redundant chunk has occurred.

It is easy to see that large D2-rings consisting of many edge nodes can effectively eliminate all duplicate chunks from their data flows, thus achieving high storage space efficiency[2]. Since edge nodes are often geo-distributed, however, the network cost resulting from large D2-rings can be significant; i.e., larger network resources are spent to access the distributed hash values stored on peer nodes which may be located in other edge clouds. To jointly minimize the storage space and network cost in distributed deduplication, we consider each edge node i as a data source that generates data chunks at a rate of R_i chunks per second. Note that, since we send the data chunks to the central cloud, on the D2 rings (which are at the edge), we are only concerned with the transient storage space for a certain time window, which serves as a proxy for the WAN bandwidth usage to send chunks to the central cloud.

To model the spatial and temporal correlation both within each data source and between different data sources, we assume that each chunk generated by source i is randomly drawn from K disjoint chunk pools, which is denoted by $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K$, with known probabilities $p_{i1}, p_{i2}, \ldots, p_{iK}$. For example, $\mathcal{C}_1$ represents chunks typical for Windows OS, $\mathcal{C}_2$ for Linux, and $\mathcal{C}_3$ for chunks shared by the two systems due to common applications. We further assume that each chunk of source i is independently generated by randomly selecting a chunk pool with probabilities $\{p_{ik},\ \forall k\}$ and then choosing a

2. We use storage cost and storage space interchangeably in this paper.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2022.3155357, IEEE Transactions on Network Science and Engineering

chunk within the selected pool with a uniform distribution. We denote the probability vector $P_i = [p_{i1}, p_{i2}, \ldots, p_{ik}]$ as the *characteristic vector* of source i, which quantifies the statistics of its data flow.

In our model, data generated by correlated sources have the same probability of selecting chunks from the K chunk pools, resulting in higher redundancy. These probabilities can be obtained by data source profiling and/or estimated through meta data (Sec. 3.1). If a set of data sources (i.e., edge nodes) are clustered into a single D2-ring, which is denoted by $\mathcal{P}$ where $\mathcal{P} \subseteq \mathcal{N}$, their data flows are jointly deduplicated. Let $\Omega(\mathcal{P})$ be the expected overall deduplication ratio of all sources in $\mathcal{P}$, i.e., the original data size divided by the deduplicated storage size. The expected storage space required for D2-ring $\mathcal{P}$ during an interval of T seconds is given by

$$U(\mathcal{P}) = \frac{1}{\Omega(\mathcal{P})} \cdot \sum_{i \in \mathcal{P}} R_i T \tag{1}$$

where R_i is the data rate of source i.

While more edge nodes in a single D2-ring increases the chance of finding redundant chunks, it also incurs higher network cost during deduplication, because as the D2-ring size increases, a higher fraction of chunk hash values are stored on non-local edge nodes, resulting in higher network cost for hash lookup when new data chunks arrive. Let γ be the (chunk hash) replication factor in the D2-ring, i.e., each unique chunk hash is stored on γ distinct edge nodes. We consider a D2-ring $\mathcal{P}$ that has size $|\mathcal{P}|$. When chunk hashes are uniformly distributed on edge nodes in the D2-ring $\mathcal{P}$ (e.g., using a distributed hash table), the probability of a non-local hash lookup for any incoming data chunk is $1 - \gamma/|\mathcal{P}|$. Let v_{ij} be the network cost of a non-local hash lookup from node i to node j; e.g., it can be measured by the necessary bandwidth or network delay of the non-local hash lookup. The total network cost for deduplication in the D2-ring $\mathcal{P}$ in an interval of T seconds is thus

$$V(\mathcal{P}) = \sum_{i: i \in \mathcal{P}} \sum_{j: j \neq i, j \in \mathcal{P}} v_{ij} \frac{R_i T \left(1 - \gamma/|\mathcal{P}|\right)}{|\mathcal{P}| - 1}, \tag{2}$$

where each non-local hash lookup has equal probability $1/(|\mathcal{P}| - 1)$ to be processed by peer edge nodes $\{j : j \neq i, j \in \mathcal{P}\}$ in the D2-ring.

Our goal is to partition the edge nodes $\mathcal{N}$ into M disjoint D2-rings, i.e., $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_M$ satisfying $\cup_s \mathcal{P}_s = \mathcal{N}$, to jointly minimize the total storage space $\sum_s U(\mathcal{P}_s)$ and network cost $\sum_s V(\mathcal{P}_s)$. Let α be a tradeoff factor quantifying the relative importance of network cost to storage space, i.e., each unit network cost is equivalent to the cost of α units of storage space increment. The joint Storage and Network Optimization in Distributed Deduplication (SNOD2) is as follows:

$$\text{minimize} \quad \sum_s U(\mathcal{P}_s) + \alpha \sum_s V(\mathcal{P}_s) \tag{3}$$

$$\text{s.t.} \quad U(\mathcal{P}_s) = \frac{1}{\Omega(\mathcal{P})} \cdot \sum_{i \in \mathcal{P}_s} R_i T,$$

$$V(\mathcal{P}_s) = \sum_{i \in \mathcal{P}_s} \sum_{j \neq i, j \in \mathcal{P}_s} \frac{v_{ij} R_i T \left(1 - \gamma/|\mathcal{P}_s|\right)}{|\mathcal{P}_s| - 1}$$

$$var. \quad \mathcal{P}_s, \ \forall s, \tag{4}$$

where $\mathcal{P}_s, \ \forall s$ forms a disjoint partition of the edge nodes. By partitioning the edge nodes into collaborative D2-rings and performing deduplication within each D2-ring, this SNOD2 optimization will allow us to minimize the network cost due to distributed index structure lookup, while maintaining a competitive deduplication ratio.

3 EF-DEDUP SOLUTION

In this section, we first quantify the storage space efficiency (i.e., deduplication ratio function $\Omega(\mathcal{P}_s)$) for a given partition $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_M$. Then, we present a novel technique to estimate the characteristic vector of data sources, which is required to formulate and solve SNOD2. The technique has significant practical implications even outside the realm of this paper because it provides an analytical model to estimate chunk distribution functions of arbitrary data sources by sampling just a few files.

Then, we show that SNOD2 under our system model is NP hard and propose a greedy algorithm to solve SNOD2 where all partitions (i.e., D2-rings) have equal sizes (for better load-balancing). The algorithm can be proven optimal when the number of disjoint chunk pools $K = 2$, and has a guaranteed competitive ratio when $K > 2$. Then, we also develop an arbitrary-partitioning algorithms for SNOD2 without any constraints of partition sizes, by leveraging matching heuristics.

3.1 Estimating Source Characteristic Vectors

To solve SNOD2, we first need to quantify the deduplication ratio of any given partition based on the chunk pools and characteristic vectors ($P_i = [p_{i1}, p_{i2}, \ldots, p_{ik}]$ for source i) that best represent the sources.

Theorem 1. *For a set of data sources in $\mathcal{P}_s$, which are generated from K disjoint chunk pools $\{\mathcal{C}_k\}$ with characteristic vectors $\{P_i, \ \forall i\}$, the deduplication ratio of D2-ring $\mathcal{P}_s$ is given by*

$$\Omega(\mathcal{P}_s) = \frac{\sum_{i \in \mathcal{P}_s} R_i T}{\sum_{k=1}^{K} s_k \left(1 - \prod_{i \in \mathcal{P}_s} g_{ik}\right)}, \quad \text{where } g_{ik} = \left(1 - p_{ik}/s_k\right)^{R_i T}. \tag{5}$$

The key question to address is: **For an unknown set of sources, how do we know the chunk pools and the characteristic vectors that best represent the sources?**

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2022.3155357, IEEE Transactions on Network Science and Engineering

Algorithm 1 Estimating Source Characteristic Vectors

Input: A set of source files $\mathcal{R}$, where f_i files are sampled randomly from each source i $\forall i$, and an error threshold.
foreach subset $\mathcal{A} \subseteq \mathcal{R}$
  Measure ground truth, i.e., real-dedup-ratio $\hat{\Omega}$ ($\mathcal{A}$);
end
do
  foreach subset $\mathcal{A} \subseteq \mathcal{R}$
    Calculate model-dedup-ratio Ω ($\mathcal{A}$) (Theorem 1);
  end
  Distance = $(\sum_{\mathcal{A}} |\Omega_(\mathcal{A}) - \hat{\Omega}(\mathcal{A})|^2 / 2^{|\mathcal{N}|})$;
  Update K, s_k, and p_{ik} by predefined stepsize;
while (Distance > error threshold);
Output: the number of chunk pools: K; the size of chunk pools: $\{s_k, \forall k\}$, and characteristic vectors: $\{P_i = [p_{i,1}, \ldots, p_{i,K}], \forall i\}$.

In algorithm 1, we address this problem by first sampling a few files at random from each source and exhaustively searching across all possible values of the parameters that we need to obtain in our model, viz. the number of chunk pools, the size of each chunk pool and the chunk distribution or characteristic vector of each source.

To find the parameters that best fit our model to a given set of data sources, we first obtain ground truth using a small set files that are sampled from the sources. For each possible partition of the files, we measure the total deduplication ratio using standard tools (e.g., duperemove) and compare these empirical values to the analytical deduplication ratio given in Theorem 1. Then, the optimal modeling parameters are obtained by minimizing the difference between analytical results and ground truth. In practice, this search algorithm can terminates once the mean square distance is smaller than a given error threshold, as summarized in Algorithm 1.

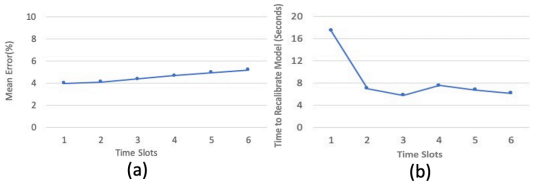


Fig. 2. Model recalibration for time-varying data: (a) The MSE of estimation error remains to be small for time-varying data, (b) The mode can be quickly recalibrated with respect to data change.

To validate our model, we sample files across the 3 real-world datasets (described in Sec. 5), estimate their characteristic vectors using Algorithm 1 and compute the deduplication ratio using our analytic model described in Theorem 1.

The 10 files from source 1 (i.e., dataset 1 files), 10 files from source 2 (i.e., dataset 2 files), and 10 files from source 3(dataset 3 files) will form $\frac{30 \times 29}{2} = 435$ combinations. For each combination, we measure the real deduplication ratio. Next, we set up our model for three chunk pools ($K = 3$) whose sizes s_k for $k = 1, 2, 3$, and probabilities p_{i1}, p_{i2}, p_{i3} for source $i = 1, 2, 3$ of the 3 sources, are to be determined through model fitting. To search for the optimal parameters, we increase each s_i to 200,000 with a step size of 100, and search each probability value p_{ik} from 0 to 1 with a step size of 0.01. Larger chunk pools and smaller step size can further improve the performance but will also extend the search space. For all of the combination of samples, we calculate the MSE (Mean Square Error) between estimated deduplication ratios and the real ones. The MSE is less than 0.32, and the average estimation error across combinations is less than 4%. We will use the model if the mean square error is small enough. In our experiments, the estimation model by using 10% of sample files could achieve estimation error small enough.

Files from the same source tend to have large deduplication ratios due to temporal correlation, and the similarity pattern across files are relatively stable. Thus, we could compute our model up front and continue to recalibrate it whenever data characteristics change . The recalibration is low cost as it only needs to improve an existing model with new data samples and does not require re-optimization from scratch. In Fig.3.1(a), we consider time-varying data sources and show the estimation error between real deduplication ratio of files and the estimated values over time. For the accelerometer records and the traffic video frames datasets, timestamps are readily available and used for model recalibration. We also feed different Linux kernel files to our model as if they are generated in different time slots. The estimation accuracy of our proposed model decreases slightly due to time-varying data characteristics, but the mean error keeps below 5%.Although the first time to estimate the characteristic vectors require optimizing the model from scratch (still in less than 4 minutes), to recalibrate and update the characteristic vectors for adapting time-varying data is very fast. This is because the search will start from previous model states, and only small changes in the modeling parameters are needed to update the model and maintain low estimation error. In Fig.3.1(b), it shows that the search only takes less than 10 seconds to update the similarity model when the file of sources in Fig.3.1(a) change over time.

Designing better heuristics for this estimation (e.g., through intelligent sampling) and proving their accuracy is an exciting avenue for future research, with wide applicability. For example, it can help guide how the chunk sizes should be selected for deduplication (e.g., to choose the chunk size that minimizes estimation error) or what should be maintained in the dedupication cache (e.g., to maintain the chunks that appear with higher probability in the chunk pools).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2022.3155357, IEEE Transactions on Network Science and Engineering

3.2 SNOD2 is NP-Hard

To show that the SNOD2 is NP-Hard, we first apply Theorem 1 and rewrite SNOD2 as follows:

$$\text{minimize} \quad \sum_s U(\mathcal{P}_s) + \alpha \sum_s V(\mathcal{P}_s) \tag{6}$$

$$\text{s.t.} \quad U(\mathcal{P}_s) = \sum_{k=1}^{K} s_k \left(1 - \prod_{i \in \mathcal{P}_s} g_{ik}\right), \tag{7}$$

$$g_{ik} = (1 - p_{ik}/s_k)^{R_i T}, \ \forall i, k \tag{8}$$

$$V(\mathcal{P}_s) = \sum_{i: i \in \mathcal{P}_s} \sum_{j: j \neq i, j \in \mathcal{P}} v_{ij} \frac{R_i T \left(1 - \gamma/|\mathcal{P}_s|\right)}{|\mathcal{P}_s| - 1} \tag{9}$$

$$var. \quad \mathcal{P}_s, \ \forall s.$$

Theorem 2. *SNOD2 is NP hard.*

Proof. We show that the *minimum k-cut* problem (which is known to be NP hard when k is an input variable [15]) can be transformed into a version of SNOD2 with zero network cost.

Consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with an assignment of weights to the edges, denoted by $w(v_1, v_2)$ for any edge $(v_1, v_2) \in \mathcal{E}$. The minimum k-cut problem partitions vertices in $\mathcal{V}$ into k disjoint sets, $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$, while minimizing the sum of removed edge weights:

$$\sum_{(v_1, v_2) \in \mathcal{E}} w(v_1, v_2) \cdot \mathbf{1}_{\{\nexists \mathcal{P}_s : v_1 \in \mathcal{P}_s, v_2 \in \mathcal{P}_s\}}, \tag{10}$$

where $\mathbf{1}_{\{\nexists \mathcal{P}_s : v_1 \in \mathcal{P}_s, v_2 \in \mathcal{P}_s\}}$ is an indicator function that is equal to 0 if vertices v_1, v_2 are in the same partition, and 1 otherwise.

We construct a version of SNOD2 where each vertex in $\mathcal{V}$ is considered as a data source and each edge in $\mathcal{E}$ corresponds to a chunk pool. Let $N = |\mathcal{V}|$ be the number of sources/vertices. Then, for each edge $(v_1, v_2) \in \mathcal{E}$, we construct a separate chunk pool for the two data sources v_1 and v_2. The chunk pool is labeled by $k = v_1 N + v_2$ (for $v_1 < v_2$) and is assigned a size $s_k = w(v_1, v_2)/(1 - c)^2$, for some constant $c \in (0, 1)$. Thus, there is a one-to-one correspondence between the edges and the chunk pools. Next, let $d(v_1)$ be the degree of a vertex $v_1 \in \mathcal{V}$. We construct a characteristic vector for data source v_1 by setting $p_{v_1,k} = 1/d(v_1)$ if vertex v_1 is an endpoint of edge k (i.e., satisfying $k = v_1 N + v_2$), and $p_{v_1,k} = 0$ otherwise. For each source v_1, we choose a data rate $R_{v_1} = \log(c)/[T \cdot \log(1 - p_{ik}/s_k)]$ for some positive T and the same constant c. Finally, all network costs are assumed to be zero.

Now we prove that SNOD2 finds a disjoint partition of $\mathcal{V}$ to minimize the same objective function in (10). Using (6) and (7), the optimization objective of SNOD2 becomes

$$\begin{aligned} &\sum_s \sum_k s_k \left(1 - \prod_{i \in \mathcal{P}_s} g_{ik}\right) \\ &= \sum_k s_k \sum_s \left(1 - \prod_{i \in \mathcal{P}_s} g_{ik}\right) \\ &= \sum_k s_k \left(N - \sum_s \prod_{i \in \mathcal{P}_s} g_{ik}\right) \end{aligned} \tag{11}$$

Next, according to SNOD2, if the edge corresponding to chunk pool k does not contain source/vertex i, we have $p_{ik} = 0$, which means $g_{ik} = 1$ due to (8). Therefore, for a given k in the last summation of (11), $g_{ik} \neq 1$ if and only if $i = v_1$ or $i = v_2$ for edge (v_1, v_2) satisfying $k = v_1 N + v_2$. In this case, it is easy to see that $g_{v_1 k} = (1 - p_{v_1 k}/s_k)^{R_{v_1} T} = c$ by plugging $R_{v_1} = \log(c)/[T \cdot \log(1 - p_{ik}/s_k)]$ into (8). Thus, for any vertex set $\mathcal{P}_x$, if it contains both vertex v_1 and v_2, we have $\prod_{i \in \mathcal{P}_x} g_{ik} = g_{v_1 k} g_{v_2 k} = c^2$. If it contains only vertex v_1 or vertex v_2, we have $\prod_{i \in \mathcal{P}_x} g_{ik} = c$; and if it does not contain v_1 or v_2, $\prod_{i \in \mathcal{P}_x} g_{ik} = 1$.Then, for edge k, we consider two cases:

$$\begin{aligned} \sum_s \prod_{i \in \mathcal{P}_s} g_{ik} &= \Big(\sum_{i \neq v1, v2} \prod_{i \neq v1, v2} g_{ik}\Big) + g_{v_1 k} + g_{v_2 k} \\ &= N - 2 + 2c, \text{ if } \nexists \mathcal{P}_s : v_1 \in \mathcal{P}_s, v_2 \in \mathcal{P}_s, \\ \sum_s \prod_{i \in \mathcal{P}_s} g_{ik} &= \Big(\sum_{i \neq v_1, v_2} \prod_{i \neq v_1, v_2} g_{ik}\Big) + g_{v_1 k} g_{v_2 k} \\ &= N - 1 + c^2, \text{ otherwise,} \end{aligned} \tag{12}$$

which is consolidated using indicator function $\mathbf{1}_{\{\exists \mathcal{P}_s : v_1 \in \mathcal{P}_s, v_2 \in \mathcal{P}_s\}}$, i.e.,

$$\sum_s \prod_{i \in \mathcal{P}_s} g_{ik} = N - 1 + c^2 - (1 - c)^2 \cdot \mathbf{1}_{\{\nexists \mathcal{P}_s : v_1 \in \mathcal{P}_s, v_2 \in \mathcal{P}_s\}}$$

Plugging this into the last step of (11), we have

$$\begin{aligned} &\sum_s \sum_k s_k \left(1 - \prod_{i \in \mathcal{P}_s} g_{ik}\right) \\ &= \sum_k s_k (1 - c^2) + \sum_k s_k (1 - c)^2 \mathbf{1}_{\{\exists \mathcal{P}_s : v_1 \in \mathcal{P}_s, v_2 \in \mathcal{P}_s\}} \\ &= \sum_k s_k (1 - c^2) + \sum_k w(v_1, v_2) \mathbf{1}_{\{\exists \mathcal{P}_s : v_1 \in \mathcal{P}_s, v_2 \in \mathcal{P}_s\}} \end{aligned}$$

where we used $s_k = w(v_1, v_2)/(1 - c)^2$ in the last step, and v_1, v_2 are the two vertices belonging the edge corresponding to chunk pool $k = v_1 N + v_2$. Notice that $\sum_k s_k (1 - c^2)$ is a constant not affected by the partitioning, and that the summation over index k is the same as v_1, v_2 (due to one-to-one correspondence between chunk pools and edges in our construction). We conclude that any solution to SNOD2 solves the minimum k-cut problem, which implies that SNOD2 is also NP hard. □

3.3 Our Proposed Solution to SNOD2

3.3.1 A Greedy Algorithm for Balanced Partitioning

We consider a balanced version of SNOD2 where for a given set of $N = Z \cdot M$ edge nodes[3], we partition them into M disjoint D2-rings, each containing an equal number of edge nodes $|\mathcal{P}_s| = Z \; \forall s$. Using the same proof in Theorem 1, it is easy to show that balanced SNOD2 is also NP hard, since balanced k-cut problem is known to be NP hard. We develop a greedy algorithm for the problem and analyze its competitive ratio.

Algorithm 2 Greedy Balanced Partitioning (GBP) Algorithm

```
foreach cluster i
    foreach Z-node partition P_j in remaining nodes
        Calculate the aggregate cost U(P_j) + α · V(P_j);
        Find P_min with the smallest aggregate cost;
    end
    P_i = P_min;
    Remove Z nodes of partition P_i from remaining nodes;
end
```

In the proposed algorithm for balanced SNOD2, we iteratively select Z edge nodes for a partition $\mathcal{P}$ that has the smallest aggregate cost $U(\mathcal{P}) + \alpha \cdot V(\mathcal{P})$ among all remaining nodes, and group them into a new D2-ring. The process continues until there are no edge nodes left and all M D2-rings are created. It is easy to see that this algorithm requires computing the aggregate cost of N-choose-Z possible partitions satisfying $\mathcal{P} \in \mathcal{N}$ and $|\mathcal{P}| = Z$. In the algorithm summarized in Algorithm 2, we sort all N-choose-Z partitions in an ascending order with respect to their aggregate costs. In each iteration, we choose a remaining available partition with the smallest aggregate cost. Once a partition $\mathcal{P}$ is selected, the nodes in $\mathcal{P}$ are removed from further consideration. The algorithm solves balanced SNOD2 in M iterations, with a computation complexity $o([N\text{-choose-}Z] \cdot \log\ [N\text{-choose-}Z])$ due to sorting of the possible partitions. For data sources that generate temporally-correlated files (like system backups) or geographically-correlated files (like traffic cameras with overlapping views), our algorithm could achieve large deduplication ratio and high throughput.

Theorem 3. *For equal network costs, the proposed algorithm for balanced SNOD2 is optimal when $K = 2$, and its required storage space achieves a competitive ratio $1 + \left(\frac{S}{U^*} - 1\right)\left(1 - \frac{1}{Z}\right)$ when $K > 2$, where U^* is the optimal storage space and $S = \sum_k s_k$ is the total size of all chunk pools.*

Theorem 3 implies that for $K > 2$, the competitive ratio of our greedy algorithm approaches 1 as S/U^* decreases, i.e., in a heavy-traffic system with more data sources and chunks.

3. If the number of edge nodes N is not an integer multiple of M, we can add dummy nodes with zero data rate to satisfy this condition.

4 Design and Implementation

In this section, we present the novel system design for EF-dedup, in which we maintain the deduplication index structure across edge nodes (that may belong to different edge clouds). We then present the implementation details of our system wherein we modify duperemove [14], a commonly used open source deduplication tool, to store chunk hashes in Cassandra [13], a popular distributed key-value store.

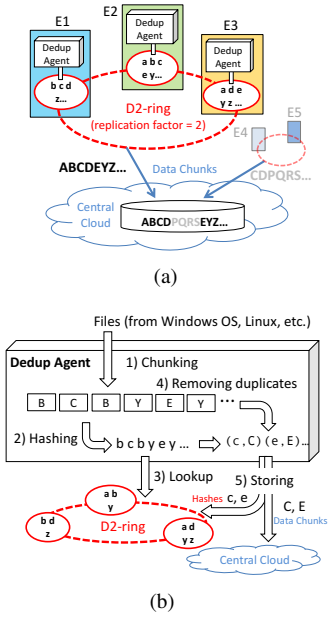


Fig. 3. EF-dedup system architecture with D2-rings traversing edge nodes that could belong to different edge clouds.

In Fig. 3 we depict our system architecture, with an example of five edge nodes E1 $\sim$ E5 that are clustered into two independent D2-rings. Our choice of Cassandra was motivated by important features required to implement the D2-ring. First, Cassandra has the notion of a ring to evenly distribute all data in the system. System administrators can allocate arbitrary number of edge nodes to each ring to create an independent name-space. This concept is very well-aligned with our definition of D2-ring. We store hash values generated by the modified version of duperemove in multiple Cassandra rings. Second, Cassandra supports data replication, and consequently the chunk hashes will be available in multiple edge nodes.

Each edge node runs our *Dedup Agent*, which performs the task of deduplicating the input files by maintaining the chunk hashes in the D2-ring's Cassandra cluster. Each Cassandra ring maintains multiple copies of chunk hashes depending on their replication factor. In Fig. 3(a) data chunk A's hash (= a) is stored in two nodes (i.e., E2 and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2022.3155357, IEEE Transactions on Network Science and Engineering

E3) and data chunk B's hash (= b) is also stored in two nodes (i.e., E1 and E2) because their replication factor is two. While more copies of hashes in Cassandra enables the *Dedup Agent* to perform local lookups for hashes, this also increases the storage needed for the hashes.

Fig. 3(b) shows how our *Dedup Agent* deduplicates files generated from diverse sources such as Windows, Linux, and so forth. After splitting files into smaller chunks, the *Dedup Agent* computes the hash value of each chunk. The *Dedup Agent* then performs a lookup to determine if this hash value is present in the Cassandra cluster and *only if it is not present* it adds this new hash to the cluster and sends the data chunk corresponding to this hash to the central cloud. For example, in Fig. 3(b), only unique chunks (i.e., C, E) across the files are sent to the central cloud.

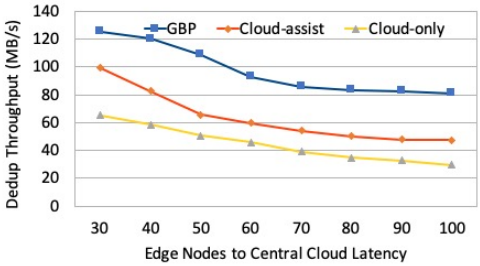
To implement *Dedup Agent*, we modified duperemove, which is a user-space application program for finding duplicate extents (contiguous storage areas in a file system). We use 4KB chunk size in this paper with SHA-256 hash values as indexes, which only lead to a small overhead for storing the index tables at edge nodes.

To do the deduplication, we modified duperemove (approximately 1200 lines of additional code) to deal with files. We take 3 files for example. File 1 has 1100 bytes, file 2 has 1200 bytes, file 3 has 800 bytes, and we set 500 bytes as the largest chunk size. First, we use duperemove to check file 1 and 2, and find they share 900 identical contents in the middle. Then file 1 will be divided into 4 chunks- chunk 1 (100 bytes), chunk 2 (500 bytes, duplicate), chunk 3 (400 bytes, duplicate) and chunk 4 (100 bytes). File 2 will be divided into 4 chunks- chunk 5 (100 bytes), chunk 2 (500 bytes, duplicate), chunk 3 (400 bytes, duplicate), and chunk 6 (200 bytes). For file 3, we compare it with former 6 chunks, and find file 3 also has identical chunk 2 in the middle. Then file 3 will be divided into chunk 7(200 bytes), chunk 2(500 bytes) and chunk 8(100 bytes). Since we only store chunk 2 and chunk 3 once, we will save 1400 bytes for these 3 files. In this way, we could avoid boundary shift problem for duplication.

To store these 3 files, we maintaines 4 tables. One chunk table maintains unique chunks, timestamps, hashes of chunks, and the index of chunks. Three file tables to restore files. Each file has its table to store the sequence of chunks(table index), and the index of chunks. Then, to restore a file, we just need to find the corresponding file table, read the sequence of chunks, and pick up the chunks from the chunk table. In our experiment, to restore a 100MB file, the system uses 3 seconds as average.

Compared to the size of chunks, the size of headers (including table index, timestamps, and so on) is too small to be considered. The size of headers is less than 5% of chunks' size.

Additionally, we replaced its Sqlite [16] database with Cassandra, for reasons mentioned above. For connecting duperemove to Cassandra, we use the DataStax C/C++ driver [17].



(a) As the latency between the edge nodes and the central cloud increases, EF-dedup continues to outperform both Cloud-only and Cloud-assisted strategies; for all, there are 4 edge nodes per D2-ring and 5 D2-rings.

Fig. 4. EF-dedup vs other Cloud-based Approaches

5 Evaluation

In this section, we demonstrate the efficacy of EF-dedup through real experiments and detailed simulations. First, we illustrate the need for edge-based distributed deduplication by comparing the throughput of EF-dedup with other cloud-based approaches. EF-dedup achieves up to 67.4% improvement in throughput over a pure central cloud-based deduplication approach, while it achieves up to 133.7% improvement over a cloud-assisted approach in which the deduplication is done at the edge, but the hash lookup is done at the central cloud. Then, we show how EF-dedup achieves the appropriate trade-off between network and storage cost, wherein it achieves 20.0-62.6% lesser aggregate cost than approaches that favor one over the other. Finally, we perform extensive simulations to evaluate the performance of different flavors of EF-dedup. Section 5.1 describes the experiment setup, including testbed, datasets, and evaluation metrics. Section 5.2 presents the baseline algorithms for comparison. Section 5.3 shows the main evaluation results on network and storage tradeoff enabled by EF-Dedup. Finally, Section 5.4 conducts additional simulations for large scale experiments.

5.1 Experimental Setup

Testbed: Our edge environment consists of 20 VMs (*edge nodes*) created on a local OpenStack [18] cluster, where each VM has 4 VCPUs, 8 GB RAM and 20 GB virtual disk drive. We install the EF-dedup deduplication agent (or Dedup Agent in Fig. 3) on each VM along with different Cassandra clusters traversing the VMs based on our experiments. To compare EF-dedup with cloud-based approaches, we also set up a 4 VM cluster on Amazon EC2 (*central cloud*), where each VM has 8 VCPUs, 15GB RAM, 20 GB storage. We run our deduplication agents based on duperemove on each of them with one large Cassandra cluster traversing all the VMs. The average data transfer rate (or bandwidth) among the edge nodes measures 1.726 Gbps with average latency as 0.85ms, while the average data transfer rate between the edge nodes and the central cloud measures 0.377 Gbps with average latency as 12.2 ms. To

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2022.3155357, IEEE Transactions on Network Science and Engineering

make our experiments more representative of average wide-area and inter edge node latencies, we add 37.8ms to the edge to central cloud latency (total = 50 ms) and 4.15ms to inter edge-cloud latency (total = 5 ms) to serve as default. We use NetEm [19] to control latency in traffic among clusters. All results presented in this paper are averaged over 20 runs with the deduplication performed in parallel at all the VMs in the system.

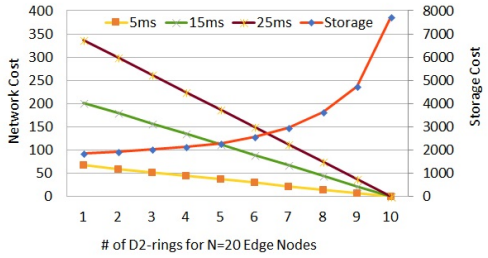
Data-Sets: We use 3 real world datasets: (1) consisting of 200 hours of accelerometer information recorded over 25 days from 5 participants [20], with each data point in the size range of 80-187MB. The dominant motion frequency of all collected traces ran in the range of 1.92-2.8 Hz, which corresponds to human walking; (2) a series of continuous frames extracted from a traffic video sequence recorded by stationary cameras [9][21]; Linux Kernel archive subversions, each of which has size in the range of 700-800 MB, which we refer to simply as "kernel dataset". For these datasets, consequent files or versions have high deduplication ratio. The mapping of datasets to nodes is random.

Evaluation Metrics: We compare different solutions primarily on two metrics: *dedup ratio* is defined as an original data size divided by the deduplicated size, while *dedup throughput* is measured by the amount of data processed per second by each edge node.

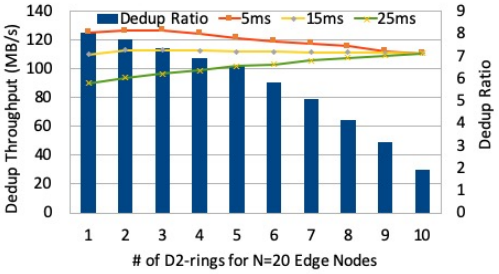
5.2 Baselines for comparison

We compare our proposed GBP algorithm with two cloud-based approaches: (1) *Cloud-only* approach, where raw data is sent from edge nodes to the central cloud for deduplication. (2) *Cloud-assisted* approach where the index structure for deduplication is maintained in the central cloud and the edge nodes after splitting the files into chunks, look up the hash of the chunks remotely and only send those chunks that are not already present in the cloud. In all these experiments, since we know the similarity patterns of the input dataset, we implement a simple version of EF-dedup in which we manually ensure that within each D2-ring the edge nodes have relatively similar data.

In Fig. 4(a), we fix the number of D2-rings to 5 (each ring containing 4 edge nodes) and vary the latency between the edge and the central cloud. While all three strategies are negatively impacted by additional latency, EF-dedup still achieves significant throughput improvement, and its lead over the other strategies is maintained even under high latency. On average, EF-dedup has 67.8% and 136.23% more dedup throughput than Cloud-assist and Cloud-only. In particular, it performs even better when cloud latency is lower than 50ms since EF-dedup can efficiently utilize resources available at peer edge nodes, while the percentage-wise improvement becomes smaller. In EF-dedup, hash look-ups for distributed deduplication only generate network traffic between edge nodes, making it more resilient to adverse network conditions between the edge and the central cloud.



(a) For a fixed number of edge nodes, as the number of D2-rings increase, there are fewer edge nodes per ring, leading to decreased deduplication and increased storage cost. However, the network cost decreases since there are fewer rings traversing edge nodes across edge-clouds



(b) The same experiment as the adjacent graph illustrates that while dedup ratio behaves in the expected inverse manner w.r.t storage cost, the dedup throughput behaves in the expected inverse manner w.r.t network cost only for inter edge-cloud latencies $\leq$ 15 ms.

Fig. 5. Inversely Proportional Network and Storage Cost in EF-dedup

5.3 Evaluating Network and Storage Tradeoff

In this section, we highlight the inverse relationship between network cost and storage cost in our edge-based collaborative deduplication solution. In Fig. 5(a) we show how the number of edge nodes in each D2-ring and the latencies among the edge node clusters affect the network and storage cost defined in the SNOD2 problem (with $\alpha = 0.1$). To capture the notion of edge-clouds, we group the twenty edge nodes into 10 equal-size groups, each group representing an edge-cloud, wherein the latency among edge nodes across the edge-clouds is increased using NetEm(by default 5ms), while the latency among edge nodes within an edge-cloud is left unchanged 0.85 ms. We vary the number of D2-rings used by EF-dedup on top of these edge-clouds to vary the number of nodes in each D2-ring from 20 to 2. For example, a data point with (# D2-rings = 2) corresponds to two D2 rings each containing ten edge nodes, while (# D2-rings = 6) corresponds to 4 D2-rings each containing 3 edge nodes, and 2 D2-rings each containing 4 edge nodes. Clearly, the D2-rings may go across nodes belonging to different edge clouds and thus suffer from inter edge-cloud latency. We use the vdbench data to perform the experiments and the same version of EF-dedup used for the experiments

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2022.3155357, IEEE Transactions on Network Science and Engineering

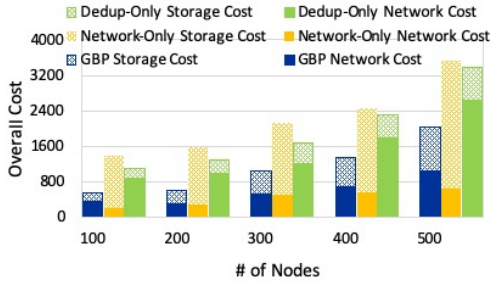
in Fig. 4.

As seen in Fig. 5(a), the storage cost increases with increasing number of rings (fewer edge nodes per ring) due to decreased opportunities to find redundant chunks. But the network cost increases with larger rings, since there will be more chances that edge nodes belonging to the same D2-ring will traverse edge-clouds, leading to higher latency for hash look-ups. In Fig. 5(b), we illustrate the subtle effect of the same experiment on the dedup throughput (dedup ratio behaves in the expected inverse manner to storage cost). When inter edge-cloud latency is less than or equal to 15ms, with larger ring size, the higher chance for redundant chunks cancel out the negative influence of larger network cost and hence results in good throughput. But above 15ms, the network cost outweighs the gains in redundancy and the throughput decreases with increasing ring size. By quantifying this tradeoff in our EF-dedup framework, a system administrate can choose the optimal operating point based on the specific design objectives. For instance, for 25ms delay, if one wants to maximize deduplication ratio with a minimum throughput requirement of 100MB/s, then Fig.5 shows that the optimal operating point is to have 5 D2-rings (and an average of 4 nodes per D2-ring) with a deduplication ratio of 6.5.

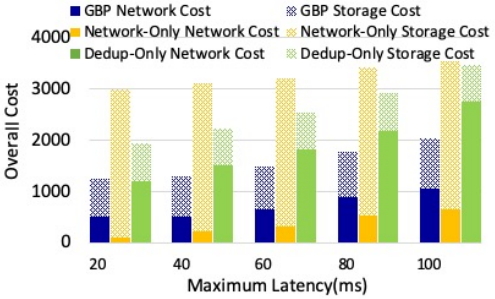
5.4 Simulations for Large Scale Experiments

In this section, we perform simulations to compare different variants of the EF-dedup algorithms to evaluate their performance at larger scale (100 edge nodes) and using data source models obtained from 3 datasets. To simulate the network cost, we draw the latency among the edge nodes from a uniform distribution between 0 to 100 ms. To simulate the storage cost, we use the characteristic vectors and chunk pools obtained from datasets described in section 3.1. We implement GBP algorithm and compare it with the arbitrary and greedily partitioned Dedup-only and Network-only algorithms. Here, Dedup-only algorithm only considers the storage space cost to partition D2-rings, and Network-only algorithm only considers the network cost to partition D2-rings.

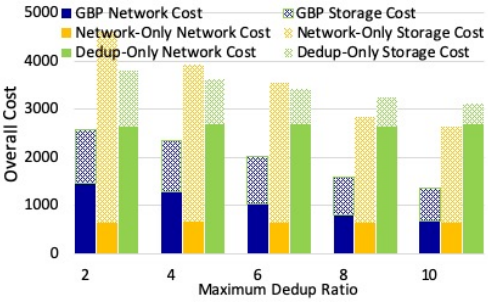
Here, we present simulation results for 0 to 500 edge nodes with inter node latency drawn from a uniform distribution between 0 to 100 ms for the second dataset. The results for the first dataset are similar. Fig. 6(a) shows the aggregate, network and storage cost (as defined in SNOD2) of the algorithms with increasing number of edge nodes. GBP uses 100 unbalanced D2 rings. It outperforms other solutions in the trade off performance as captured by the aggregate cost especially for larger number of edge nodes since there are more options to find optimal partitions. For 500 nodes GBP has 43.15% & 45.3% less aggregate costs than Network-Only and Dedup-Only algorithms respectively. For GBP, we divide the 500 edge nodes into 100 D2-rings each containing 5 edge nodes. It has similar results as GBP - 40.01% & 43.01% less aggregate costs than other algorithms.



(a) GBP has the smallest aggregate cost, which is more pronounced as the number of edge nodes increase, since there are more opportunities to find optimal solutions.



(b) GBP has the smallest aggregate cost, no matter whether the latency among nodes are high or low.



(c) GBP has the smallest aggregate cost, no matter whether the data among nodes are similar or not.

Fig. 6. Comparing different variants of EF-dedup.

Then, we test the performance of GBP for different network status, from uniform 0-20ms latency among nodes to 0-100ms seconds among nodes. Fig. 6(b) shows GBP always has the smallest aggregate costs, beating Network-only and Dedup-only algorithm by much lower storage cost and network cost respectively. GBP has $43.15-59.38\%$&$39.47-45.29\%$ less aggregate costs than Network-Only and Dedup-Only algorithms respectively. When latencies are low, GBP pays more attention on storage optimization, and could has storage cost nearly as low as Dedup-only algorithm. When latencies are high, GBP prioritizes network cost, and its storage cost increases in order to turn down network cost.

We also present simulation results for different data similarities among nodes. The data deduplication ratio

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2022.3155357, IEEE Transactions on Network Science and Engineering

among nodes are from 0-2 to 0-10. Fig. 6(b) shows GBP always has the smallest aggregate costs by perfectly balancing network and storage cost. When data similarities are high, GBP could has network cost approaching Network-only algorithm. When data similarities are low, the network cost of GBP will be higher to exchange much less storage cost. GBP has similar results as GBP.

6 Related Work

Vast amount of related work exists in the area of data deduplication and inspired our work. Unlike existing body of work which often focused on reducing local memory usage or local disk accesses, EF-dedup proposes a way to collaboratively perform efficient hash lookup in an emerging edge network environment (Sec. 1). EF-dedup trades off increased inter-edge network traffic for reduced storage cost, and thereby saving significant amount WAN usage towards the central cloud. Moreover, our system model for overall deduplication process is unique, which is significantly different from the existing similarity-based deduplication systems. In this section, we discuss a body of work directly related to EF-dedup technique. More comprehensive overview can be found in [6].

Clustered Deduplication: To the best of our knowledge, the closest body of work is the notion of clustered deduplication, which involves multiple machines for detecting duplicated data. Many systems first perform coarse-grained deduplication with bigger processing units, distribute them to multiple machines using DHT or other load balancing algorithms, and the servers involved in the process will perform more fine-grained deduplication to further optimize the deduplication ratio. As an example, HYDRAstor [22] first deduplicates incoming data using larger chunk size, e.g., 64KB, and then sends intermediate results to multiple servers so that they can perform more fine-grained deduplication. $\sum$-dedup [23] exploits similarity for distributing super-chunks and utilizes locality to achieve faster index structure lookups. Other works in this area [24], [25] take a similar high-level approach. These works mainly focused on enabling data deduplication for secondary storage typically in a powerful data center environment, while EF-dedup's main focus is more on how to best utilize edge network resources. Broadly speaking, these techniques are complementary to our work since we can apply more advanced, and computationally expensive, deduplication the data arrive at the central cloud.

Source Deduplication for Cloud Storage: The idea of deduplicating large volumes of data from the client- or source-side to save either or both network and storage cost has been extensively studied before. For instance, similar to our EF-dedup, AA-Dedupe [26] and SAFE [27] proposed source deduplication systems for optimizing cloud backup services, motivated by scarce WAN bandwidth. AA-Dedupe clusters incoming data per application type which allows them to achieve good deduplication ratio. SAFE utilizes both global file level redundancy and local chunk-level redundancy to achieve better deduplication ratio. More references and discussions can be found in [6]. Unlike EF-dedup, most work in this category does not makes multiple sources collaborate with one another, which is a main differentiating factor with the proposed EF-deduptechnique.

Similarity-aware Deduplication: Another body of work considers data similarity to aid deduplication process [28], [29], [23], [30], which share some common grounds with our EF-dedup technique. SAP [28] explored trade-offs between access-efficiency (or throughput) and space efficiency and provides an algorithm to partition data across nodes based on computing the full pair-wise similarities across files. Aronovich [30], on the other hand, focused on efficient techniques to finding similar chunks using smaller signatures representative of the chunk. EF-dedup uses an entirely novel technique to model the similarity across data sources, based on estimating the probability distribution of the sources, by sampling a few of their chunks. Further it explores a unique trade-off in the edge between network cost and storage. The systems such as SiLo [29] and $\sum$-dedup leverage both data similarity and locality. Both techniques use data similarity to decide more coarse-grained unit of processing and then apply a data mining technique to identify data locality, e.g., similar to the observation made in [7]. However, unlike EF-dedup, SiLo's focus lies in optimizing local memory utilization and $\sum$-dedup optimizes dedicated backup infrastructure in a single datacenter.

Optimizing Local System Resources: Other works on improving chunk hashing and indexing process improve memory utilization of a single physical host, consequently reducing the needs for accessing slow disk drives. The goal has been achieved in various contexts, for instance, based on locality [7], [31], data similarity [28], [32] and/or relying on faster media such as SSDs [33], [34]. These techniques can co-exist with EF-dedup technique so as to boost individual edge node's deduplication performance (dedup-ratio).

7 Conclusion

Data deduplication is a prime candidate for edge processing, since we can exploit the geographical correlation of data closer to the sources to suppress duplicated data that will otherwise be sent to the central cloud, thereby saving significant amount of WAN bandwidth. We present an edge-facilitated deduplication technique, EF-dedup, in which we partition edge nodes into independent deduplication clusters (also referred as rings in the paper), carefully balancing the deduplication ratio and the deduplication throughput. We formulate a joint storage and network optimization problem with a novel data model to capture data similarities across sources. Further, we implement EF-dedup based on an efficient heuristic to this NP-Hard problem and confirm its efficacy with experiments on real-world datasets across an OpenStack-based local cloud and AWS-based central cloud.

For future work, we will consider online optimizations in a dynamic environment and a joint optimization with file retrievals. We also wish to investigate deduplication with variable chunk sizes and to improve the performance of our source estimation algorithm through techniques like locality sensitive hashing [35] and provide a library of common chunk pools by profiling publicly available datasets. Furthermore, we want to explore efficient data sampling techniques, tradeoffs with computation cycles and power, and the problem of storing the actual data chunks at the edge and data access in reading phase, which will bring forth new trade-offs based on the popularity of the data and the size of storage.

REFERENCES

[1] "ATT is Reinventing the Cloud Through Edge Computing," http://about.att.com/story/reinventing_the_cloud_through_edge_computing.html.

[2] "Verizon's cloud-in-a-box pushes the edge with OpenStack," https://siliconangle.com/blog/2017/07/17/verizons-cloud-box-pushes-edges-openstack-openstacksummit.

[3] Y. Li, Y. Chen, T. Lan, and G. Venkataramani, "Mobiqor: Pushing the envelope of mobile edge computing via quality-of-result optimization," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 1261–1270.

[4] "Cisco Global Cloud Index: Forecast and Methodology, 2016-2021 White Paper," https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html, 2018.

[5] "IDC Directions 2017: IoT Forecast, 5G & Related Sessions," http://techblog.comsoc.org/2017/03/04/idc-directions-2017-iot-forecast-related-sessions/, 2017.

[6] W. Xia, H. Jiang, D. Feng, F. Douglis, P. Shilane, Y. Hua, M. Fu, Y. Zhang, and Y. Zhou, "A comprehensive study of the past, present, and future of data deduplication," *Proceedings of the IEEE*, vol. 104, no. 9, pp. 1681–1710, 2016.

[7] B. Zhu, K. Li, and R. H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system." in *Fast*, vol. 8, 2008, pp. 1–14.

[8] "Avamar: Deduplication Backup Software and System," https://www.emc.com/data-protection/avamar.htm.

[9] H. Yan, X. Li, Y. Wang, and C. Jia, "Centralized duplicate removal video storage system with privacy preservation in iot," *Sensors*, vol. 18, no. 6, p. 1814, 2018.

[10] Y. Zhang, Y. Wu, and G. Yang, "Droplet: A distributed solution of data deduplication," in *Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing*. IEEE Computer Society, 2012, pp. 114–121.

[11] T. Süß, T. Kaya, M. Mäsker, and A. Brinkmann, "Deduplication analyses of multimedia system images," in {*USENIX*} *Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[12] S. L. Li, T. Lan, B. Balasubramanian, M.-R. Ra, H. W. Lee, and R. K. Panta, "Ef-dedup - enabling collaborative data deduplication at the network edge," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019.

[13] A. Lakshman and P. Malik, "Cassandra: structured storage system on a p2p network," in *Proceedings of the 28th ACM symposium on Principles of distributed computing*, ser. PODC '09. New York, NY, USA: ACM, 2009, pp. 5–5. [Online]. Available: http://doi.acm.org/10.1145/1582716.1582722

[14] "Duperemove," https://github.com/markfasheh/duperemove.

[15] P. Manurangsi, "Inapproximability of maximum edge biclique, maximum balanced biclique and minimum k-cut from the small set expansion hypothesis," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 80. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[16] "Sqlite," https://www.sqlite.org.

[17] "Datastax c/c++ driver for apache cassandra," https://github.com/datastax/cpp-driver.

[18] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, 2012.

[19] S. Hemminger *et al.*, "Network emulation with netem," in *Linux conf au*, 2005, pp. 18–23.

[20] M. Cong, K. Kim, M. Gorlatova, J. Sarik, J. Kymissis, and G. Zussman, "CRAWDAD dataset columbia/kinetic (v. 2014-05-13)," Downloaded from https://crawdad.org/columbia/kinetic/20140513/kinetic-energy, May 2014, traceset: kinetic-energy.

[21] M. Wang, W. Li, and X. Wang, "Transferring a generic pedestrian detector towards specific scenes," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3274–3281.

[22] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki, "Hydrastor: A scalable secondary storage." in *FAST*, vol. 9, 2009, pp. 197–210.

[23] Y. Fu, H. Jiang, and N. Xiao, "A scalable inline cluster deduplication framework for big data protection," in *Proceedings of the 13th international middleware conference*. Springer-Verlag New York, Inc., 2012, pp. 354–373.

[24] T. Yang, H. Jiang, D. Feng, Z. Niu, K. Zhou, and Y. Wan, "Debar: A scalable high-performance de-duplication storage system for backup and archiving," in *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 1–12.

[25] W. Dong, F. Douglis, K. Li, R. H. Patterson, S. Reddy, and P. Shilane, "Tradeoffs in scalable data routing for deduplication clusters." in *FAST*, vol. 11, 2011, pp. 15–29.

[26] Y. Fu, H. Jiang, N. Xiao, L. Tian, and F. Liu, "Aa-dedupe: An application-aware source deduplication approach for cloud backup services in the personal computing environment," in *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*. IEEE, 2011, pp. 112–120.

[27] Y. Tan, H. Jiang, E. H.-M. Sha, Z. Yan, and D. Feng, "Safe: A source deduplication framework for efficient cloud backup services," *Journal of Signal Processing Systems*, vol. 72, no. 3, pp. 209–228, 2013.

[28] B. Balasubramanian, T. Lan, and M. Chiang, "Sap: Similarity-aware partitioning for efficient cloud storage," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 592–600.

[29] W. Xia, H. Jiang, D. Feng, and Y. Hua, "Silo: A similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput." in *USENIX annual technical conference*, 2011, pp. 26–30.

[30] L. Aronovich, R. Asher, E. Bachmat, H. Bitner, M. Hirsch, and S. T. Klein, "The design of a similarity based deduplication system," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*. ACM, 2009, p. 6.

[31] D. Meister, J. Kaiser, and A. Brinkmann, "Block locality caching for data deduplication," in *Proceedings of the 6th International Systems and Storage Conference*. ACM, 2013, p. 15.

[32] D. Bhagwat, K. Eshghi, D. D. Long, and M. Lillibridge, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009. MASCOTS'09. IEEE International Symposium on*. IEEE, 2009, pp. 1–9.

[33] D. Meister and A. Brinkmann, "dedupv1: Improving deduplication throughput using solid state drives (ssd)," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–6.

[34] B. K. Debnath, S. Sengupta, and J. Li, "Chunkstash: Speeding up inline storage deduplication using flash memory." in *USENIX annual technical conference*, 2010, pp. 1–16.

[35] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of the 25th International Conference on Very Large Data Bases*, ser. VLDB '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 518–529. [Online]. Available: http://dl.acm.org/citation.cfm?id=645925.671516

APPENDIX

A.1 Proof of Theorem 1

Proof. It is easy to see that the original data flow size is $\sum_{i\in\mathcal{P}_s} R_i T$ for an interval of T seconds. Without loss of generality, we consider a data chunk in pool $\mathcal{C}_k$. Based on our data flow construction model in Sec. 2, this chunk is selected when source i generates a new chunk, with probability p_{ik}/s_k where s_k is the size of chunk pool k. The probability that the chunk is never selected by source i during an interval T is given by $g_{ik} = (1 - p_{ik}/s_k)^{R_i T}$, since a total of $R_i T$ chunks are generated. Then, $\prod_{i\in\mathcal{P}_s} g_{ik}$ is the probability that a chunk in $\mathcal{C}_k$ is never selected by any source during T. Since all chunks in pool $\mathcal{C}_k$ are selected with the same probability, the expected number of distinct chunks drawn from $\mathcal{C}_k$ by all sources is thus $s_k\left(1 - \prod_{i\in\mathcal{P}_s} g_{ik}\right)$, whose summation over all chunk pools $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K$ yield the total required storage space after deduplication. □

A.2 Proof of Theorem 3

Proof. When network costs are equal, SNOD2 reduces to a minimization of required storage space $\sum_s \sum_k s_k \left(1 - \prod_{i\in\mathcal{P}_s} g_{ik}\right)$ over balanced partitions $\mathcal{P}_1, \ldots, \mathcal{P}_M$. Since $\sum_s \sum_k s_k$ is constant, the problem boils down to maximizing

$$\sum_s \sum_k s_k \prod_{i\in\mathcal{P}_s} g_{ik} = \sum_k s_k \left(\sum_s \prod_{i\in\mathcal{P}_s} g_{ik}\right). \quad \text{(A1)}$$

Proof of optimality when $K = 2$. Consider the problem of partitioning $N = MZ$ positive numbers $g_{1k}, g_{2k}, \ldots, g_{Nk}$ into M equal-size subsets $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_M$, to maximize the sum of products, $\sum_s \prod_{i\in\mathcal{P}_s} g_{ik}$, which is the last term in (A1). It can be shown that the optimal solution to this problem is to sort the numbers in descending (or ascending) order (denoted by $g^{\downarrow}_{1k}, g^{\downarrow}_{2k}, \ldots, g^{\downarrow}_{Nk}$) and then group every M adjacent numbers into each partition, i.e., $\mathcal{P}_s = \{g^{\downarrow}_{sN-N+1,k}, g^{\downarrow}_{sN-N+2,k}, \ldots, g^{\downarrow}_{sN,k}\}$, for $s = 1, \ldots, M$.

We prove this by contradiction. Assume that $\sum_s \prod_{i\in\mathcal{P}_s} g_{ik}$ is maximized by some optimal partition, in which there exist $x_1, y_1 \in \mathcal{P}_1$ and $x_2, y_2 \in \mathcal{P}_2$, but $g_{x_1k} > g_{y_1k}$ and $g_{x_2k} < g_{y_2k}$ (i.e., these numbers are not sorted). Without loss of generality, we assume $\prod_{i\in\mathcal{P}_1/\{x_1,y_1\}} g_{ik} \geq \prod_{i\in\mathcal{P}_2/\{x_2,y_2\}} g_{ik}$, which are the products excluding $g_{x_1k}, g_{y_1k}, g_{x_2k}, g_{y_2k}$. It is easy to see that if we swap $y_1 \in \mathcal{P}_1$ and $y_2 \in \mathcal{P}_2$, it yields a strictly higher objective value, since $g_{x_1k} g_{y_2k} \prod_{i\in\mathcal{P}_1/\{x_1,y_1\}} g_{ik} + g_{x_2k} g_{y_1k} \prod_{i\in\mathcal{P}_2/\{x_2,y_2\}} g_{ik} > g_{x_1k} g_{y_1k} \prod_{i\in\mathcal{P}_1/\{x_1,y_1\}} g_{ik} + g_{x_2k} g_{y_2k} \prod_{i\in\mathcal{P}_1/\{x_2,y_2\}} g_{ik}$. This contradicts with the optimality of partitions $\mathcal{P}_1, \mathcal{P}_2$ in our assumption.

Notice that when $K = 2$, we have $g_{i1} + g_{i2} = 1\ \forall i$. Therefore, our proposed greedy algorithm that partitions the edge nodes by sorting $g_{11}, g_{21}, \ldots, g_{N1}$ (for $k = 1$) in a descending order automatically sorts $g_{12} = 1 - g_{11}, \ldots, g_{N2} = 1 - g_{N1}$ in an ascending order. The solution simultaneously optimizes $\sum_s \prod_{i\in\mathcal{P}_s} g_{ik}$ for both $k = 1$ and $k = 2$, which is an optimal solution maximizing (A1) and thus balanced SNOD2.

Proof of competitive ratio when $K > 2$**.** Let $\phi(\mathcal{P}_s) = \sum_k s_k \prod_{i\in\mathcal{P}_s} g_{ik}\ \ \forall s$. We consider the maximization of (A1), which is $\sum_s \phi(\mathcal{P}_s)$. We use $\mathcal{P}_1, \ldots, \mathcal{P}_M$ to denote the feasible partition obtained by our proposed greedy algorithm, and $\mathcal{P}^*_1, \ldots, \mathcal{P}^*_M$ for the optimal partition. We first show that

$$\sum_s \phi(\mathcal{P}_s) \geq \frac{1}{Z} \sum_s \phi(\mathcal{P}^*_s), \quad \text{(A2)}$$

where Z is the size of each partition. We denote the optimal objective value above by $\Gamma(\mathcal{N}) = \sum_s \phi(\mathcal{P}^*_s)$ for $\mathcal{N} = \cup_s \mathcal{P}^*_s$.

We prove the result by induction. Consider $M \leq Z$. Our greedy algorithm first finds a group of Z sources, $\mathcal{P}_1$, which has the largest sum of product, i.e., $\phi(\mathcal{P}_1) \geq \phi(\mathcal{P})$ for any $\phi(\mathcal{P}) \subset \mathcal{N}$. It is easy to see that

$$\sum_s \phi(\mathcal{P}_s) \geq \phi(\mathcal{P}_1) \geq \sum_{s=1}^{M} \phi(\mathcal{P}^*_s)/M \geq \sum_{s=1}^{M} \phi(\mathcal{P}^*_s)/Z, \quad \text{(A3)}$$

where we use the facts that $\mathcal{P}_1$ has the largest sum of product and that $M \leq Z$. Now suppose that the result holds for all $M \leq M_0$. We consider $M = M_0 + 1$. The Z nodes belonging to $\mathcal{P}_1$ would be distributed in $x \leq Z$ distinct partitions in the optimal solution, i.e., $\mathcal{P}^*_1, \ldots, \mathcal{P}^*_x$ without loss of generality. It is easy to see that $\mathcal{P}^*_1, \ldots, \mathcal{P}^*_x$ offers an optimal partitioning for a subset of nodes $\cup_{s=1}^{x} \mathcal{P}^*_s$. Similarly, $\mathcal{P}^*_{x+1}, \ldots, \mathcal{P}^*_M$ is optimal for a subset of nodes $\cup_{s=x+1}^{M} \mathcal{P}^*_s$. Then, we have

$$\begin{aligned}
\sum_{s=1}^{M} \phi(\mathcal{P}_s) &= \phi(\mathcal{P}_1) + \sum_{s=2}^{M} \phi(\mathcal{P}_s) \\
&\geq \frac{1}{x} \sum_{s=1}^{x} \phi(\mathcal{P}^*_s) + \frac{1}{Z} \Gamma(\cup_{s=2}^{M} \mathcal{P}_s) \\
&\geq \frac{1}{x} \sum_{s=1}^{x} \phi(\mathcal{P}^*_s) + \frac{1}{Z} \Gamma(\cup_{s=x+1}^{M} \mathcal{P}^*_s) \\
&= \frac{1}{x} \sum_{s=1}^{x} \phi(\mathcal{P}^*_s) + \frac{1}{Z} \sum_{s=x+1}^{M} \phi(\mathcal{P}^*_s) \\
&\geq \frac{1}{Z} \sum_{s=1}^{M} \phi(\mathcal{P}^*_s). \quad \text{(A4)}
\end{aligned}$$

The second step follows from the fact that $\phi(\mathcal{P}_1) \geq \phi(\mathcal{P}^*_s)$ for any s due to our greedy algorithm, and from the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSE.2022.3155357, IEEE Transactions on Network Science and Engineering

induction assumption that $\sum_{s=2}^{M} \phi(\mathcal{P}_s) \geq \Gamma(\cup_{s=2}^{M} \mathcal{P}_s)/Z$ for any set of $(M-1)Z = M_0 Z$ nodes, i.e., $\cup_{s=2}^{M} \mathcal{P}_s$. $\Gamma(\mathcal{N})$ denotes the maximum objective value by optimally partitioning nodes in $\mathcal{N}$. The third step holds because $\cup_{s=x+1}^{M} \mathcal{P}_s^* \subset \cup_{s=2}^{M} \mathcal{P}_s$, and thus a higher maximum objective Γ value is always achieved when more nodes are added to the system. The fourth step uses the fact that the partition $\mathcal{P}_{x+1}^*, \ldots, \mathcal{P}_M^*$ is optimal for a subset of nodes $\cup_{s=x+1}^{M} \mathcal{P}_s^*$. Finally, the last step uses $x \leq Z$. Therefore, the induction assumption holds for $M = M_0 + 1$.

Using $S = \sum_s s_k$ and the optimal storage space U^*, we obtain the competitive ratio:

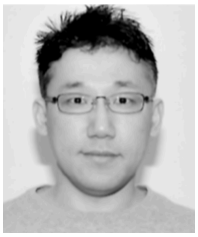
$$\frac{S - (S - U^*)/Z}{U^*} = 1 + \left(\frac{S}{U^*} - 1\right)\left(1 - \frac{1}{Z}\right). \quad \text{(A5)}$$

This completes the proof of our theorem. □

Shijing Li received the BE degree from the Beijing University of Posts and Telecommunications, in 2014. She is working toward the PhD degree at the George Washington University. Her research focus includes distributed storage systems, traffic scheduling, and edge computing.

Tian Lan received the B.A.Sc. degree from the Tsinghua University, China in 2003, the M.A.Sc. degree from the University of Toronto, Canada, in 2005, and the Ph.D. degree from the Princeton University in 2010. Dr. Lan is currently a full Professor of Electrical and Computer Engineering at the George Washington University. His research interests include network optimization, algorithms, and machine learning. Dr. Lan received the SecureComm Best Paper Award in 2019, the SEAS Faculty Recognition Award at GWU in 2018, the Hegarty Faculty Innovation Award at GWU in 2017, AT&T VURI Award in 2014, the INFOCOM Best Paper Award in 2012, the IEEE GLOBECOM Best Paper Award in 2009, and the IEEE Signal Processing Society Best Paper Award in 2008.

Bharath Balasubramanian received the BE degree in electronics from Mumbai University in 2004, and the MS and PhD degrees in computer engineering from the University of Texas at Austin in 2007 and 2012, respectively. Currently, he is a Senior Software Engineer at Google. His areas of interest include: concurrent and distributed algorithms, fault tolerant distributed systems, distributed storage, and distributed debugging.



Hee Won Lee received a Ph.D. degree in Computer Science from North Carolina State University in May 2015. He received a B.E. in Electrical Engineering from Korea University in 2002, and a Master of Software Engineering from Carnegie Mellon University in 2005. During 2002-2009, he worked for KT Corporation as a Technical Member of Staff. He is currently employed as a Principal Member of Technical Staff at AT&T Labs Research. His primary research interest is in networking and storage systems.

Moo-Ryong Ra received the PhD degree from Computer Science Department, University of Southern California. He is a principal inventive scientist with the AT&T Labs Research. He is interested in systems and networking. In AT&T, more focus is given to the following areas: software defined storage for cloud platform/infrastructure in the context of AT&T integrated cloud and edge cloud, video storage and delivery, RDMA networking for next generation storage/memory architecture. Before joining AT&T, he had built several interesting cloud-enabled mobile sensing systems to better understand the interaction between smart mobile devices and the cloud infrastructure.

Panta Krishna Rejesh received the PhD degree in electrical and computer engineering from Purdue University. He is a principal inventive scientist with the AT&T Labs-Research. His research interests include cloud computing, Big Data, storage systems, distributed systems, wireless networks, mobile systems, and sensor networks.