2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)

EF-dedup: Enabling Collaborative Data Deduplication at the Network Edge

Shijing Li*, Tian Lan*, Bharath Balasubramanian**, Moo-Ryong Ra**, Hee Won Lee**,Rajesh Panta**
*George Washington University, **AT&T Research Lab
{shijing, tlan}@gwu.edu, {bharathb, mra, knowpd, rpanta}@research.att.com

***Abstract*—The advent of IoT and edge computing will lead to massive amounts of data that need to be collected and transmitted to online storage systems. To address this problem, we push data deduplication to the network edge. Specifically, we propose a new technique for collaborative edge-facilitated deduplication (EF-dedup), wherein we partition the resource-constrained edge nodes into disjoint clusters, maintain a deduplication index structure for each cluster using a distributed key-value store and perform decentralized deduplication within those clusters. This is a challenging partitioning problem that addresses 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 address this challenge by first formulating an optimization problem to partition the edge nodes, considering both the data similarities across the nodes and the inter-node network cost. We prove that the problem is NP-Hard, provide bounded heuristics to solve it and build a prototype EF-dedup system. Our experiments on EF-dedup, performed on edge nodes in AT&T research lab and a central cloud at AWS, demonstrate that EF-dedup achieves 38.3∼118.5% better deduplication throughput than sole cloud-based techniques and achieves 43.4-60.2% lesser aggregate cost in terms of the network-storage trade-off as compared to approaches that solely favor one over the other.**

***Index Terms*—Deduplication, Edge Computing, Edge Networks, Distributed Storage, Cloud Storage**

I. Introduction

The emergence of IoT and edge computing will result in smart mobile devices, connected cars, sensors etc. generating large volumes of data. Large telecommunication companies are building edge cloud infrastructures [1], [2], [3] to support IoT data management applications, virtualized RAN for 5G, augmented/virtual reality applications and virtual networks like consumer provider edge routers (CPE). According to recent market analysis results, the data stored in the edge-facing IoT devices is expected to reach 5.9 ZB by 2021 [4] and crucially, only 43% of such data will be processed in edge clouds [5]. The remaining data will be sent to the central cloud for further storage and processing. In Fig. 1, for example, nodes in edge clouds receive data from IoT devices (smart phones/sensors) and these edge nodes in turn send these massive amounts of data to the central cloud.

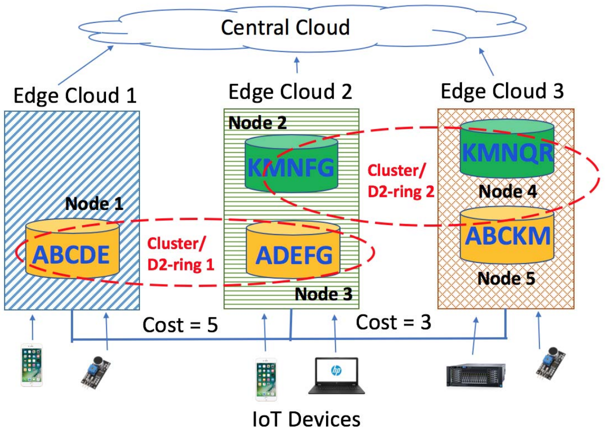


Fig. 1: An example of collaborative, edge-centric deduplication and the associated network-storage trade-off.

Data deduplication, which is a well studied [6], commercially applied technique [7], [8], has been adopted to optimize storage space in modern datacenters. Deduplication 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 work loads 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 make the novel observation that to minimize the data sent from the edge to the central cloud, deduplication should be pushed 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. 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 latencies. 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. Hence, the main technical problem we solve in this paper is: *How can the decentralized edge nodes collaborate to perform distributed deduplication, with the objective of optimizing both required storage space and network cost?*

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 $\sim$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.

In this paper, we address these challenges and propose a novel technique for collaborative Edge-Facilitated Deduplication (EF-dedup) wherein we partition edge nodes into disjoint clusters (henceforth, *D2-rings*[2]) and perform decentralized deduplication within each D2-ring, such that the solution is optimal according to the network-storage trade-off described above. We maintain an index structure across the edge nodes of each D2-ring in a fault-tolerant distributed key-value store, which consumes very little resources per node and is robust to unreliable nodes and networks. Specifically, 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. II). We propose a new time-varying hierarchical estimation model to capture similarity across correlated data sources, using chunk pools and characteristic vectors. Our model is validated using real-world datasets with an average estimation error less than 4% and offline prediction time less than $\sim$4 minutes (Sec. III-A).
- We prove that the formulated problem is NP-Hard and provide efficient heuristics to the problem with bounded approximate performance ratio (Sec. III) .
- We present a system for edge-facilitated deduplication, EF-dedup, where we maintain the index structure of each deduplication D2-ring in a key-value store across the nodes of the ring. We implement EF-dedup by modifying duperemove [12] to use Cassandra [13] for its index structure (Sec. IV).

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

[2]Distributed-Deduplication-rings. In this paper, "cluster", "partition", and "D2-ring" refer to the same entity.

- We perform experiments and simulations on AWS and OpenStack using both IoT and edge-related real-world datasets (Sec. V), which confirm both the high throughput and deduplication ratio achieved by EF-dedup. Specifically, EF-dedup achieves 67.4$\sim$133.7% better deduplication throughput than sole cloud-based techniques, and also 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.

II. System Model and Problem Formulation

We consider a set of N distributed edge nodes, e.g., VMs in cloudlets/fog/edge clouds, denoted by $\mathcal{N} = \{1, 2, \dots, N\}$. 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 transfered 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. 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 equal-size 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 $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_K$, with known probabilities $p_{i1}, p_{i2}, \dots, p_{iK}$. 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 chunk within the selected pool with a uniform distribution. We denote the probability vector $P_i = [p_{i1}, p_{i2}, \dots, p_{ik}]$ 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. III-A). 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:

$$\begin{aligned}
\text{minimize} \quad & \sum_s U(\mathcal{P}_s) + \alpha \sum_s V(\mathcal{P}_s) && (3)\\
\text{s.t.} \quad & U(\mathcal{P}_s) = \frac{1}{\Omega(\mathcal{P})} \cdot \sum_{i \in \mathcal{P}_s} R_i T, \\
& V(\mathcal{P}_s) = \sum_{i \in \mathcal{P}_s} \sum_{j \neq i, j \in \mathcal{P}_s} \frac{v_{ij} R_i T \left(1 - \gamma/|\mathcal{P}_s|\right)}{|\mathcal{P}_s| - 1} \\
var. \quad & \mathcal{P}_s, \ \forall s, && (4)
\end{aligned}$$

where $\mathcal{P}_s, \ \forall s$ forms a disjoint partition of the edge nodes.

III. EF-dedup Solution

In this section, we first quantify 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 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 arbitrary-partitioning algorithms for SNOD2 without any constraints on partition sizes, by leveraging matching heuristics.

A. 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*

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

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. II, 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. □

Algorithm 1 Estimating Source Characteristic Vectors

```
Input: (i) A set of source files $\mathcal{R}_t$ periodically sampled across points of time ($t = 1, 2, \ldots$) and randomly from each source $i$ $\forall i$, and (ii) an error threshold.
foreach time point $t = 1, 2, \ldots$
  foreach subset $\mathcal{A} \subseteq \mathcal{R}_t$
    Measure ground truth: real-dedup-ratio $\hat{\Omega}$ ($\mathcal{A}$);
  end
  do
    foreach subset $\mathcal{A} \subseteq \mathcal{R}_t$
      Calculate model-dedup-ratio $\Omega$ ($\mathcal{A}$) in Theorem 1, using characteristic vectors for previous time slot $t-1$;
    end
    MSE = $(\sum_{\mathcal{A}} |\Omega(\mathcal{A}) - \hat{\Omega}(\mathcal{A})|^2)/2^{|\mathcal{N}|}$;
    Update $K$, $s_k$, and $p_{ik}$ by predefined stepsize;
  while (MSE > error threshold);
  Output the number of chunk pools: $K_t$; the size of chunk pools: $\{s_{kt}, \ \forall k\}$, and characteristic vectors: $\{P_{it} = [p_{i1t}, \ldots, p_{iKt}], \ \forall i\}$;
end
```

A key problem that we need to address is that for an unknown set of sources, how do we know the chunk pools and the characteristic vectors that best represent the sources? In algorithm 1, we address this problem by tracking the similarity of random samples of each source over time where for each time point we exhaustively search across all possible values of parameters that we need for 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 deduplication 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 Algorithm 1, this search terminates once the mean square distance is smaller than a given error threshold.

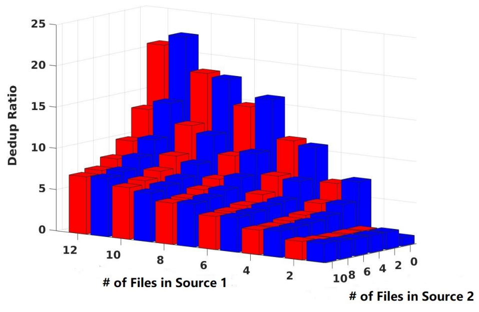


Fig. 2: The difference on real dedup ratio and the estimated one is less than 4%.

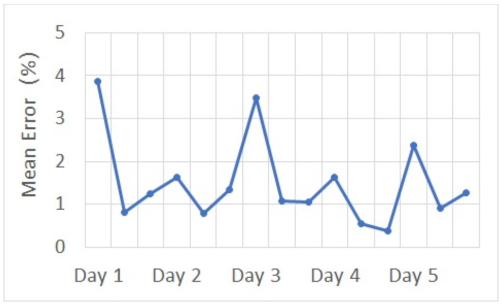


Fig. 3: The difference between the real and estimated dedup ratio (less than 4%) generally decreases across time as the estimation improves.

To validate the feasibility of our model, we sample files across the real-world accelerometer IoT data sets, estimate their characteristic vectors using Algorithm 1 and compute the deduplication ratio using our analytic model described in Theorem 1. Specifically, we sample 3 times within a day's data, and begin with the first and second samples of files as source 1 and 2 (datasets described in Sec.V). We show the real deduplication ratios in blue bars and the estimated deduplication ratios in red bars in Fig.2.

The $0th, 2nd, \ldots, 10th$ files from source 1 and $0th, 2nd, \ldots, 10th$ files from source 2 will form 6×6 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 $i = 1, 2$ of the two 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 36 combination of samples, we calculate the MSE (Mean Square Error) between estimated deduplication ratios and the real ones.

In Fig.2, the MSE is less than 0.3, and the average estimation error across combinations is less than 4%. We will use the model if the mean square error is small enough. Then, we use the second and third sample files of the first day as the input sources 1 and 2, but this time we begin with previous characteristic vectors when searching, and the searching ends extremely quickly in several seconds with even smaller errors. Figure 3 shows that our algorithm can yield a set of feasible modeling parameters and the mean error is less than 4%, which means on average the estimated deduplication ratio is only 4% less than the real one. In fact, for successive time slots, the search will have much smaller errors and will get faster.

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 pick 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).

B. *SNOD2 is NP-Hard*

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

$$\begin{aligned}
\text{minimize} \quad & \sum_s U(\mathcal{P}_s) + \alpha \sum_s V(\mathcal{P}_s) && (6)\\
\text{s.t.} \quad & U(\mathcal{P}_s) = \sum_{k=1}^{K} s_k \left(1 - \prod_{i \in \mathcal{P}_s} g_{ik}\right), && (7)\\
& g_{ik} = (1 - p_{ik}/s_k)^{R_i T}, \ \forall i, k && (8)\\
& V(\mathcal{P}_s) = \sum_{i: i \in \mathcal{P}_s} \sum_{j: j \neq i, j \in \mathcal{P}} v_{ij} \frac{R_i T \left(1 - \gamma/|\mathcal{P}_s|\right)}{|\mathcal{P}_s| - 1} && (9)\\
\textit{var.} \quad & \mathcal{P}_s, \ \forall s.
\end{aligned}$$

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 [14]) 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:

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

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

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

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:

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

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. □

C. Our Proposed Solution to SNOD2

In algorithm 2, we present a smart, efficient heuristic for SNOD2 with no constraints on D2-ring size, that iteratively selects a remaining edge node with the smallest cost increment and places it into one of M existing D2-rings. It begins with M empty D2-rings, $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_M$, and iteratively places a remaining node $v \in \mathcal{V}$ into ring $\mathcal{P}_s$, if

$$\{v, s\} = \arg \min_{v \in \mathcal{V}, s} \quad [U(\mathcal{P}_s \cup \{v\}) + \alpha V(\mathcal{P}_s \cup \{v\}) - U(\mathcal{P}_s) - \alpha V(\mathcal{P}_s)] \,. \quad (13)$$

In other words, placing node v in D2-ring $\mathcal{P}_s$ results in the minimum cost increment. Then, we update $\mathcal{V}/\{v\} \rightarrow \mathcal{V}$ and $\mathcal{P}_s \cup \{v\} \rightarrow \mathcal{P}_s$. The process continues until all nodes are placed. It is easy to see that the greedy algorithm has a computation complexity $o(N^2 \cdot M)$.

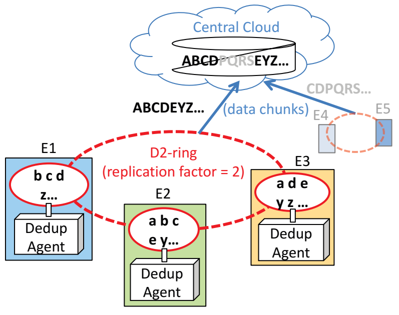
We note that the greedy algorithm can be computed efficiently via a sequence of minimum-weight matching. Starting with N nodes, we define the weight between two nodes v_1, v_2 as the aggregate cost $U(\{v_1, v_2\}) + \alpha V(\{v_1, v_2\})$ (a dummy node can be added if N is an odd number), and perform a minimum-weight matching. If we preserve the first θN matches that have the lowest weights, this step produces $N - L$ partitions, each containing 1 or 2 nodes. Next, the same minimum-weight matching is carried out on the remaining partitions (with the weight between $\mathcal{P}_1, \mathcal{P}_2$ defined again as the aggregate cost $U(\{\mathcal{P}_1 \cup \mathcal{P}_2\}) + \alpha U(\{\mathcal{P}_1 \cup \mathcal{P}_2\})$), and the first θ matches with minimum weights are preserved. This procedure reduces the number of partitions by a factor of θ in each step. The proposed algorithm converges in $o(\log_{1-\theta}(N/M))$ rounds.

Algorithm 2 Smart Partitioning (SMART) Algorithm for EF-dedup

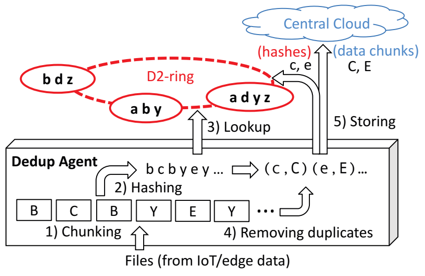
```
foreach node i
    foreach cluster $j$
        Calculate cluster aggregate cost $U(\mathcal{P}_j \cup \{v\}) + \alpha V(\mathcal{P}_j \cup \{v\}) - U(\mathcal{P}_j) - \alpha V(\mathcal{P}_j)$ ;
        Find the cluster $\mathcal{P}_{min}$ with the smallest cluster aggregate cost;
    end
    $\mathcal{P}_{min} = \mathcal{P}_{min} \cup i$;
    Remove node i from remaining nodes;
end
```

IV. Design and Implementation

Fig. 4 (a) shows our novel system architecture, with an example of five edge nodes E1 $\sim$ E5 that are clustered into two independent D2-rings, one across E1, E2, E3



(a)



(b)

Fig. 4: EF-dedup system architecture with D2-rings traversing edge nodes that could belong to different edge clouds. Each D2-ring is implemented as a Cassandra cluster storing the hashes evenly across the D2-ring nodes.

and the other across E4 and E5. In a D2-ring, each edge node runs our *Dedup Agent*. The Dedup Agent is a version of duperemove [12], a commonly used open source deduplication tool, that we modified (approximately 1200 lines of additional code) to store chunk hashes in Cassandra, a popular key-value store, which is deployed across all the nodes in a ring. Each Cassandra ring maintains multiple copies of chunk hashes depending on their replication factor, to make hashes available in multiple edge nodes. This use of Cassandra enables us to spread the storage load of multiple hashes across all the nodes in a ring; a crucial consideration given the resource constraints of the edge nodes. The other advantage of using Cassandra is its resilience against failures – even if an edge node is completely disconnected from others, it can still perform deduplication by consulting its local Cassandra node. Further, adding and removing nodes to the cluster is a seamless operation.

As seen in Fig. 4 (b), after splitting files into smaller chunks in each edge node, the *Dedup Agent* first computes the hash value of each chunk, then performs a lookup to determine if this hash value is present in the Cassandra cluster (i.e., D2-ring) 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. In Fig. 4 (b), for example, only unique chunks (i.e., C, E) across the files are sent to the central cloud. As we have mentioned previously, in this paper, we are concerned with the edge data that is sent to the cloud. The question of whether some of this data is also maintained locally at the edge nodes for locality/availability for the clients is orthogonal.

V. Evaluation

In this section, we present the results of extensive experiments and simulations that show the efficacy of EF-dudup over other approaches. Our edge testbed consists of 20 VMs (*edge nodes*) created on a local OpenStack cluster, where each VM has 4 VCPUs, 8 GB RAM and 20 GB virtual disk drive.

Our experimental deployment accurately reflects the description in section IV. We install our Dedup Agent on each of the 20 VMs. In different experiments, the SMART algorithm will divide these VMs into independent clusters or D2-rings that achieve the best network-storage trade-off. For each D2-ring, we deploy an independent Cassandra cluster traversing the nodes of the D2-ring. As shown in the D2-ring in Fig. 4 (b), the Cassandra cluster stores the hashes or the index data (e.g. a,b,y) across its nodes. Cassandra automatically partitions and replicates this data evenly across nodes based on its internal sharding algorithm. We use the random partitioning strategy of Cassandra with a replication factor of two to ensure resiliency and availability.

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 use NetEm [15] to control traffic latency among the edge clusters and the edge-central cloud clusters. The measured bandwidth among the edge nodes is 1.726 Gbps with average latency of 0.85 ms. The measured average bandwidth between the edge nodes and the central cloud is 0.377 Gbps with average latency of 12.2 ms. All results are averaged over 20 runs with the deduplication performed in parallel at all VMs in the system.

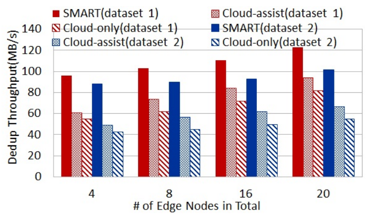
We use 2 real IoT data sets: (1) consisting of 200 hours of accelerometer information recorded over 25 days from 5 participants [16], with each data point in the size range of 80-187MB. The dominant motion frequency of all collected traces ran in the range of 1.92-2.8 Hz, which corresponds to human walking; (2) a series of continuous frames extracted from a traffic video sequence recorded by stationary cameras [9][17].

A. Comparison with Other Cloud-based Approaches

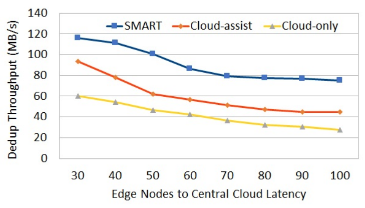
We compare SMART with a *Cloud-only* approach, where raw data is sent from edge nodes to the central cloud for deduplication and a *Cloud-assisted* approach where the index structure for deduplication is maintained in the central cloud and the edge nodes after splitting the files into chunks, look up the hash of the chunks remotely and only send those chunks that are not already present in the central cloud. In Fig. 5(a), we run SMART with 5 D2-rings, and each of ring needs not have a fixed D2-ring size. In terms of dedup throughput (data processed per second by each edge node), SMART outperforms Cloud-assisted and Cloud-only approaches by 38.3% and 59.8% (on average) for the first dataset, and 67.4% and 118.5% better for the second dataset, respectively. The Cloud-only strategy has low dedup throughput since all data is forwarded to the cloud for deduplication, thus bottlenecked by the constrained upload bandwidth. The dedup throughput of the Cloud-assisted strategy is also limited by the need for frequent message passing between edge nodes and the central cloud for hash look-ups. As the number of edge nodes increase, the throughput of SMART increases due to parallel deduplication done by more edge nodes.

In Fig. 5(b), we perform a similar experiment but vary the latency between the edge and the cloud with the first IoT dataset. While all strategies are impacted by additional latency, SMART still achieves significant throughput improvement, and its relative lead over the cloud-based strategies becomes more substantial (from 24.2% improvement of SMART vs. cloud-assisted under 30ms latency to 67.1% under 100ms latency). This is because in SMART 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. The results for the second IoT dataset are similar, where SMART has 28.1% higher throughput than the cloud-assisted algorithm and 69.8% higher throughput than the cloud-only algorithm on average. In Fig. 5(c), we see that despite the fact that SMART is upper-bounded by the cloud-based approaches in terms of space savings or dedup ratio, as we create fewer D2-rings across the twenty edge nodes (more nodes per ring) SMART quickly approaches the dedup ratio of cloud-based strategies, due to increased chances of finding chunk hashes. The trend for the second IoT dataset is similar as well.

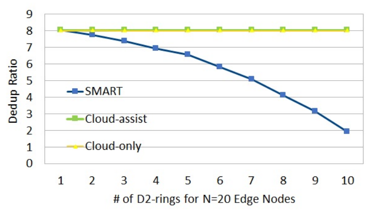
As seen above, although cloud algorithms have larger deduplication ratio they have low throughput and incur large bandwidth. In fact, with the resources at their disposal, cloud algorithms can user variable chunk sizes and other sophisticated algorithms to improve the deduplication ratio. But the main point we make is that, by using smart cluster partitioning and proper cluster size, SMART can approach acceptably high deduplication ratio with high throughput as shown in the following network-storage trade-off section.



(a)

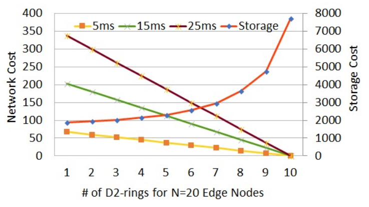


(b)

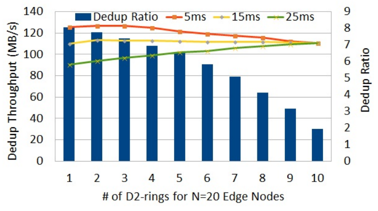


(c)

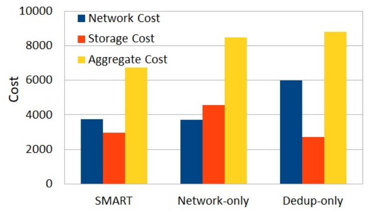
Fig. 5: Experiments confirming how EF-dedup with SMART partitioning achieves much better dedup throughput than Cloud-assisted and Cloud-only algorithms, especially for larger network size or higher latency environments without compromising too much on dedup ratio.



(a)

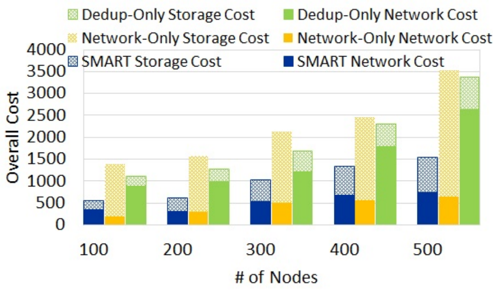


(b)

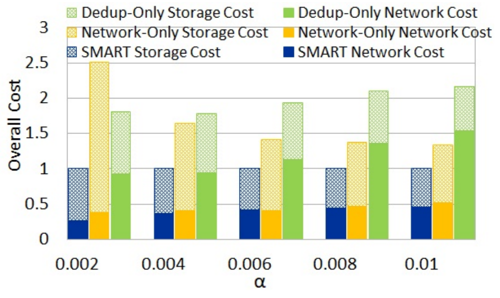


(c)

Fig. 6: Experiments illustrating how EF-dedup achieves the best network-storage trade-off. This is further emphasized by the comparison in (c) with variants that totally ignore one of these aspects for the other.



(a)



(b)

Fig. 7: Simulations on larger network size and varying values of the trade-off factor α confirm that EF-dedup outperforms other algorithms in the aggregate cost that captures the network storage trade-off.

B. Network/Storage Trade-off

In this section, we highlight the trade-off between network cost and storage cost in EF-dedup for the SMART algorithm using the IoT dataset with $\alpha = 0.1$. To capture the notion of edge-clouds, we group the twenty edge nodes into 10 geographical groups, each group representing an edge-cloud, and increase the inter-node latency across the edge-clouds using NetEm (default 5ms), while the latency among edge nodes within an edge-cloud is left unchanged 0.85 ms. Here we show the results for the first IoT dataset and the trend is similar for the second dataset.

In Fig. 6(a) we show how 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. 6(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.

In Fig. 6(c) we compare the total cost incurred by SMART as defined in Equation 3 with a flavor of EF-dedup in which we completely ignore the storage cost factor $U(\mathcal{P}_j \cup \{v\})$ (referred to as *Network-only*) in algorithm 2 and a flavor in which we completely ignore the network cost factor $\alpha V(\mathcal{P}_j \cup \{v\}) - \alpha V(\mathcal{P}_j)$ (referred to as *Dedup-only*). The Network-Only and Dedup-Only algorithm have 1.26- and 1.31-time aggregate cost as compared to SMART, which is able to intelligently trade-off network and storage cost. This result is reflected in the storage savings and throughput (figure not shown due to space constraints) wherein SMART saves 1604.36 MB storage space compared to Network-Only but has only 6.72MB/s less dedup throughput. Similarly, SMART achieves 29.16MB/s more dedup throughput than Dedup-Only, but occupies only 248.54MB more storage.

C. Simulations

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. 7(a) shows the aggregate, network and storage cost (as defined in SNOD2) of the algorithms with increasing number of edge nodes and $\alpha = 0.001$. SMART uses 20 unbalanced D2 rings. SMART 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 SMART has 43.35% & 45.49% less aggregate costs than Network-Only and Dedup-Only algorithms respectively. Fig. 7(b) confirms that as α increases, the network cost of SMART increases and storage cost decreases. For $\alpha = 0.001$, SMART outperforms other algorithms by 60.2% & 45.1% smaller aggregate cost, respectively. By tuning α we can choose the appropriate network-storage trade-off.

VI. Related Work

Prior work has considered the notion of clustered deduplication [18], [19], [20], [21], where systems like HYDRAstor [18] first perform coarse-grained deduplication with larger chunk size, and then distribute the data using DHTs or load balancers to multiple servers that perform more fine-grained deduplication. These works focus on deduplication for secondary storage in a powerful data center environment, while EF-dedup's focuses on how to best utilize edge space and network resources. These techniques are complementary to our work since we can apply more advanced and computationally expensive deduplication once the data arrives at the central cloud.

The idea of client- or source-side deduplication has been studied before [6]. For instance, AA-Dedupe [22] performs source deduplication by clustering incoming data per application type while SAFE [23] utilizes both global file level redundancy and local chunk-level redundancy at the source to achieve better deduplication ratio. Unlike EF-dedup, work in this category do not enable multiple sources (edge nodes in our case) collaborate with one another to perform distributed deduplication.

Similarity-aware deduplication has been studied in several works [24], [25], [21], [26]. SAP [24] explores trade-offs between throughput and space efficiency to partition data across nodes by computing pair-wise similarities across files. Aronovich [26], focuses on efficient techniques to finding similar chunks using smaller signatures representative of the chunk. SiLo [25] and $\sum$-dedup leverage data similarity for coarse-grained deduplication and then apply a data mining technique to identify data locality. SiLo's focus is to optimize local memory utilization and $\sum$-dedup optimizes dedicated backup infrastructure in a single datacenter. EF-dedup uses a novel technique to model similarity, based on estimating the probability distribution of the sources, by sampling a few of their chunks and uses this analysis to balance network-storage cost of edge deduplication.

VII. Conclusion

Data deduplication at the edge can exploit the geographical correlation of massive amounts of data closer to the sources, thereby suppressing duplicated data that will otherwise be sent to the central cloud. We present an edge-facilitated deduplication technique, EF-dedup, to partition edge nodes into independent deduplication clusters, 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 with bounded approximate ratio to this NP-Hard problem and validate its effectiveness with extensive experiments on real-world datasets. For future work, we wish to improve the performance of our source estimation algorithm through techniques like locality sensitive hashing [27] and provide a library of common chunk pools by profiling publicly available datasets. To make the data more reliable and save more storage space, we intend to apply erasure code to store data replicas[28], [29]. We also wish to improve the edge deduplication ratio, by using more sophisticated deduplication techniques like variable-size chunking. Finally, we wish to study some of the clustering ideas of solutions like AA-Dedupe or SAFE and adapt them intelligently and appropriately for EF-dedup in our edge environments.

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

[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] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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

[28] S. Li, T. Lan, M.-R. Ra, and R. Panta, "Joint scheduling and source selection for background traffic in erasure-coded storage," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 12, pp. 2826–2837, 2018.

[29] ——, "Background traffic optimization for meeting deadlines in data center storage," in *2016 Annual Conference on Information Science and Systems (CISS)*. IEEE, 2016, pp. 372–377.