Application-Aware Big Data Deduplication in Cloud Environment

Yinjin Fu, Nong Xiao, Hong Jiang, *Fellow, IEEE*, Guyu Hu, and Weiwei Chen

Abstract—Deduplication has become a widely deployed technology in cloud data centers to improve IT resources efficiency. However, traditional techniques face a great challenge in big data deduplication to strike a sensible tradeoff between the conflicting goals of scalable deduplication throughput and high duplicate elimination ratio. We propose *AppDedupe*, an application-aware scalable inline distributed deduplication framework in cloud environment, to meet this challenge by exploiting application awareness, data similarity and locality to optimize distributed deduplication with inter-node two-tiered data routing and intra-node application-aware deduplication. It first dispenses application data at file level with an application-aware routing to keep application locality, then assigns similar application data to the same storage node at the super-chunk granularity using a handprinting-based stateful data routing scheme to maintain high global deduplication efficiency, meanwhile balances the workload across nodes. AppDedupe builds application-aware similarity indices with super-chunk handprints to speedup the intra-node deduplication process with high efficiency. Our experimental evaluation of AppDedupe against state-of-the-art, driven by real-world datasets, demonstrates that AppDedupe achieves the highest global deduplication efficiency with a higher global deduplication effectiveness than the high-overhead and poorly scalable traditional scheme, but at an overhead only slightly higher than that of the scalable but low duplicate-elimination-ratio approaches.

Index Terms — big data deduplication, application awareness, data routing, handprinting, similarity index

1 Introduction

Recent technological advancements in cloud computing, internet of things and social network, have led to a deluge of data from distinctive domains over the past two decades. Cloud data centers are awash in digital data, easily amassing petabytes and even exabytes of information, and the complexity of data management escalates in big data. However, IDC data shows that nearly 75% of our digital world is a copy [1]. Data deduplication [2], a specialized data reduction technique widely deployed in disk-based storage systems, not only saves data storage space, power and cooling in data centers, also decreases significant administration time, operational complexity and risk of human error. It partitions large data objects into smaller parts, called chunks, represents these chunks by their fingerprints, replaces the duplicate chunks with their fingerprints after chunk fingerprint index lookup, and only transfers or stores the unique chunks for the purpose of improving communication and storage efficiency. Data deduplication has been successfully used in various application scenarios, such as backup system [1], virtual machine storage[3], primary storage [4], and WAN replication [5].

Big data deduplication is a highly scalable distributed deduplication technique to manage the data deluge under the changes in storage architecture to meet the service level agreement requirements of cloud storage. It is generally in favor of source inline deduplication design, because it can immediately identify and eliminate duplicates in datasets at the source of data generation, and hence significantly reduce physical storage capacity requirements and save network bandwidth during data transfer. It performs in a typical distributed deduplication [6], [7], [8], [9], [10], [11], [12] framework to satisfy scalable capacity and performance requirements in massive data. The framework includes inter-node data assignment from clients to multiple deduplication storage nodes by a data routing scheme, and independent intra-node redundancy suppression in individual storage nodes.

Unfortunately, this chunk-based inline distributed deduplication framework at large scales faces challenges in both inter-node and intra-node scenarios. First, for the inter-node scenario, different from those distributed deduplication with high overhead in global match query [37], [43], there is a challenge called *deduplication node information island*. It means that deduplication is only performed within individual nodes due to the communication overhead considerations, and leaves the cross-node redundancy untouched. Second, for the intra-node scenario, it suffers from the *chunk index lookup disk bottleneck*. There is a chunk index of a large dataset, which maps each chunk's fingerprint to where that chunk is stored on disk in order to identify the replicated data. It is generally too big to fit into the limited memory of a deduplication node [3], and causes the parallel deduplication performance of multiple data streams to degrade significantly due to the frequent and random disk index I/Os.

- *Yinjin Fu, Guyu Hu and Weiwei Chen are with the College of Command Information System, PLA University of Science and Technology, Nanjing, Jiangsu 210007, China. Email: yinjinfu@gmail.com, huguyu@189.cn, njcww@qq.com.*
- *Nong Xiao is with the School of Data and Computer Science, Sun Yat-Sen University, Guangzhou, Guangdong 510006, China. E-mail: xiaon6@mail.sysu.edu.cn.*
- *Hong Jiang is with the Department of Computer Science and Engineering, University of Texas at Arlington, Arlington TX 76019, USA. E-mail: hong.jiang@uta.edu.*

Manuscript received 5 Dec. 2016; revised 31 Apr. 2017.

xxxx-xxxx/0x/$xx.00 © 200x IEEE

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/TCC.2017.2710043, IEEE Transactions on Cloud Computing

There are several existing solutions that aim to tackle the above two challenges of distributed deduplication by exploiting data similarity or locality. Locality means that the chunks of a data stream will appear in approximately the same order again with a high probability. Locality-only based approaches [7], [8], [9] distribute data across deduplication servers at coarse granularity to achieve scalable deduplication throughput across the nodes by exploiting locality in data streams, but they suffer low duplicate elimination ratio due to high cross-node redundancy.Similarity in this context means that two segments [14] of a data stream or two files of a dataset share many chunks even though they arrive in a random order. The most similar stored segments or files are prefetched to deduplicate the processing segment or file in low-locality workloads by exploiting a property called logical locality [40]. Similarity-only based methods [6], [20], [30] leverage data similarity to distribute data among deduplication nodes to reduce cross-node duplication, while they also often fail to obtain good load balance and high intra-node deduplication ratio by fingerprint based mapping and allowing some duplicate chunks to be stored. In recent years, researchers [32], [33] exploit both data similarity and locality to strike a sensible tradeoff between the conflicting goals of high deduplication effectiveness and high performance scalability for distributed deduplication.

However, all these schemes are oblivious to the content and format of application files, and cannot find the redundancy in files with complex format, like video and audio files [38]. Hence, their space efficiency can be further improved by exploiting application awareness. This is a codesign of storage and application to optimize deduplication based storage systems when the deduplicated storage layer has extensive knowledge about the file structures and their access characteristics in the application layer.

As shown in Table 1, the conventional deduplication schemes always improve performance in single-node scenario or distributed scenario without considerations on application awareness. In the latest research works, application aware duplicate detection has been adopted to single-node deduplication [17], [18], [19], [22] to improve deduplication efficiency with low system overhead. In this paper, we propose AppDedupe, a scalable source inline distributed deduplication framework by leveraging application awareness, as a middleware deployable in data centers, to support big data management in cloud storage. Our solution takes aim at large-scale distributed deduplication with thousands of storage nodes in cloud datacenters which would most likely fail in the traditional distributed methods due to some of their shortcomings in terms of global deduplication ratio, single-node throughput, data skew, and communication overhead.

The main idea behind AppDedupe is to optimize distributed deduplication by exploiting application awareness, data similarity and locality in streams. More specifically, it performs two-tiered routing decision which firstly dispenses application metadata at file level with an application-aware routing to keep application locality, then assigns chunk fingerprints of intra-app similar data to the same storage node at the super-chunk [7] (i.e. consecutive smaller chunks) granularity using a handprinting-based stateful data routing scheme to maintain high global deduplication efficiency without cross-node deduplication, meanwhile balances the workload of nodes from clients. Finally, it performs application-aware deduplication in each node independently and in parallel. To reduce the overhead of resemblance detection in each node, we build an application-aware similarity index to alleviate the chunk index lookup disk bottleneck for the deduplication processes in individual nodes. The client only needs to send the unique chunks of the super-chunk to the target node, because duplicate detection process is performed in the target node before data transfer.

TABLE 1

TABLE OF RELATED WORK

Type	Application Oblivious	Application Awareness
Single-node Deduplication	DDFS [13], SiLo [16], ChunkStash [15], D2D[14],RevDedup[42]	ViDeDup [17] ADMAD [18] AA-Dedup [22] ALG-Dedupe [19]
Distributed Deduplication	ExtremeBinn [6], EMC [7], DEBAR [11], CALB [8], MAD2 [10], IBM [20], DEDIS [43], HYDRAstor [9], Symantec [30], Produck [32], Σ-Dedupe [33]	AppDedupe (This Paper)

The proposed AppDedupe distributed deduplication system has the following salient features that distinguish it from the state-of-the-art mechanisms:

- To the best of our knowledge, AppDedupe is the first research work on leveraging application awareness in the context of distributed deduplication.
- It performs two-tiered routing decision by exploiting application awareness, data similarity and locality to direct data routing from clients to deduplication storage nodes to achieve a good tradeoff between the conflicting goals of high deduplication effectiveness and low system overhead.
- It builds a global application route table and independent similarity indices with super-chunk handprints over the traditional chunk-fingerprint indexing scheme to alleviate the chunk lookup disk bottleneck for deduplication in each storage node.
- Evaluation results show that it consistently and significantly outperforms the state-of-the-art schemes in distributed deduplication efficiency by achieving high global deduplication effectiveness with balanced storage usage across the nodes and high parallel deduplication throughput at a low inter-node communication overhead.

The rest of the paper is structured as follows. Section 2 presents the necessary background to motivate the design of the AppDedupe framework. Section 3 describes the architecture of our distributed deduplication system, the

two-tiered data routing scheme and key data structures. Section 4 evaluates the AppDedupe prototype with real-world datasets and traces. Finally, Section 5 summarizes the paper.

2 Background and Motivation

In this section, we first provide the necessary background for our research to motivate our scalable inline distributed deduplication research for big data management.

2.1 Distributed Deduplication Techniques

Traditional distributed deduplication solutions, such as [9], [10], [11], [12], support exact deduplication process by routing data from clients to server nodes with the same chunk granularity as intra-node deduplication operations, and the chunk is distributed to storage nodes using a hash-bucket or distributed hash table (DHT) based stateless routing scheme. The stateless routing only uses information of the processing chunk to direct the assignment, rather than uses information about where the previous chunks were routed in stateful routing. Though they can achieve high capacity saving, these exact distributed deduplication schemes always suffer low system throughput due to weak locality in each storage node.

Extreme Binning [6] is an approximate distributed deduplication technique by exploiting file similarity. It extracts file similarity characteristic with the minimum chunk fingerprint in the file, and routes file to deduplication nodes using hashing based stateless routing. This approach limits deduplication when inter-file similarity is poor; it also suffers from increased cache misses and data skew. Similar to Extreme Binning, another file-similarity based data routing scheme is proposed by Symantec [30], but only a rough design is presented.

EMC designed both stateless and stateful versions of super-chunk routing by leveraging data locality in backup streams [7]. Its distributed deduplication scheme is built over DDFS with chunk level deduplication. The super-chunk routing is superior to using individual chunks to achieve scalable throughput while maximizing deduplication. Stateless routing is a simple and efficient way to build small deduplication cluster. However, it can't achieve high capacity saving and keep load balance for large-scale distributed deduplication. Stateful routing can achieve high duplicate elimination ratio and load balance, but it has high communication overhead for fingerprint query broadcasting.

EMC also designed a content-aware load balancing (CALB) scheme by leveraging client similarity [8]. To reduce the amount of data overlapping among different deduplication based backup appliances, it repeats a given client's backups on the same appliance, and reassigns clients to new servers only be done when the need for load balancing exceeds the overhead of data movement.

Produck tries to reduce the system overhead of stateful routing using a probabilistic method for computing the cardinality of a multiset [32]. It can significantly reduce the super-chunk assignment time, when compared to EMC stateful routing. However, its scalability is still limited due to the global query scheme with high system overhead in the coordinator node, and its memory overhead for fingerprint lookup is still very high since the data routing scheme without any help on intra-node deduplication.

TABLE 2

Comparison of Key Features among Representative Distributed Deduplication Schemes

Cluster Dedupe Scheme	Routing Granularity	Dedupe Ratio	Through put	Data Skew	Over-head
NEC HydraStor	Chunk	Medium	Low	Low	Low
Extreme Binning	File	Medium	High	Medium	Low
EMC CALB	Client	Low	High	High	Low
EMC Stateless	Super-chunk	Medium	High	Medium	Low
EMC Stateful	Super-chunk	High	Low	Low	High
Produck	Super-chunk	High	Low	Low	High
AppDedupe	Super-chunk	High	High	Low	Low

TABLE 3

Inter-App and Intra-App Redundancy Analysis

Item	Linux	Mail	VM	Web	Audio	Photo	Video
Size(GB)	160	526	313	43	153	208	366
Intra-App Saving(%)	87.8	90.5	77	47.4	33.7	28.2	11.5
Inter-App Saving(%)	0.29	0.01	0.27	0.1	0.12	0.27	0.02

Table 2 summarizes the differences among some of the typical distributed deduplication schemes, as discussed above. All these distributed deduplication mechanisms are inline methods to immediately identify and eliminate data redundancy. Our AppDedupe employs source deduplication to remove duplicate before data transfer over network rather than target deduplication in other methods. AppDedupe optimizes the data routing of distributed deduplication by exploiting application awareness, similarity and locality in data streams. In related to the existing approaches, our AppDedupe is most relevant to Produck, Extreme Binning and EMC stateful and stateless routing schemes, and it overcomes many of the weaknesses described about these schemes.

2.2 Application Difference Redundancy Analysis

In cloud data centers, the massive data comes from a large number of applications in clients. We compare the chunk fingerprints of test datasets with inter-application deduplication (inter-app) and intra-application deduplication (inter-app) using chunk-level deduplication with a fixed chunk size of 4 KB calculate the corresponding MD5 value as the chunk fingerprint in different applications, including Linux kernel source code (Linux), dataset in mail server (Mail), virtual machine images (VM), dataset in web server (Web), photo collections (Photo), music library (Audio) and movie fileset (Video). As shown in Table 3, our empirical observations and analysis reveal that the amount of data overlap among different types of applications is negligibly small due to the difference in data content and format among these applications. These results are consistent with previously published studies [22], [39]. This phenomenon motivates us to propose an application-aware data routing, which tries to route data

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/TCC.2017.2710043, IEEE Transactions on Cloud Computing

by mapping each application type to the same deduplication node. To exploit chunk-level redundancy, the best choices of chunking method and chunk size to achieve high deduplication efficiency vary with different application datasets [19]. This result is echoed in Fig. 9. After we concentrately assign the data from each application to the same storage node, the efficiency of intra-node deduplication can be significantly improved by application-aware chunking with the priori knowledge on data format in files [17], [18], [19], [38].

The application aware independent deduplication can outperform the centralized deduplication in effectiveness due to the high efficiency of application-aware chunking and negligible inter-app data overlap. We can divide the dataset into small application-affined subsets by exploiting application awareness with metadata information, such as file name or directory information in file system. Furthermore, the size of subsets is always unevenly distributed in nature. The load balance problem in the data routing decision will be analyzed in Section 4. Each application-affined subset will be assigned to a group of deduplication storage nodes rather than a single node due to the capacity limitation in each node. We will discuss the data assignment in each node group with the same application-affined data in Section 2.3.

2.3 Super-chunk Resemblance Analysis

In the hash based deduplication schemes, cryptographic hash functions, such as the MD and SHA family, are used for calculating chunk fingerprints due to their very low probability of hash collisions that renders data loss extremely unlikely. Assume that two different data chunks have different fingerprint values; we use the Jaccard index [23] as a measure of super-chunk resemblance. Let h be a cryptographic hash function, $h(S)$ denote the set of chunk fingerprints generated by h on super-chunk S. Hence, for any two super-chunks S_1 and S_2 with almost the same average chunk size, we can define their resemblance measure r according to the Jaccard index as expressed in (1).

$$r \triangleq \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} \approx \frac{|h(S_1) \cap h(S_2)|}{|h(S_1) \cup h(S_2)|} \tag{1}$$

Our similarity based data routing scheme depends on the creative feature selection on super-chunks by a handprinting technique. The selection method is based on a generalization of Broder's theorem [24]. Before we discuss the theorem, let's first introduce the min-wise independent hash functions.

Definition 1. A family of hash functions $H = \{h_i: [n] \to [n]\}$ (where $[n]=\{0, 1, \ldots, n\text{-}1\}$) is called min-wise independent if for any $X \subset [n]$ and $x \in X$, it can be formally stated as in (2), where $\Pr_{h \in H}$ denotes the probability space obtained by choosing h uniformly at random from H.

$$\Pr_{h \in H}(\min\{h(X)\} = h(x)) = \frac{1}{|X|} \tag{2}$$

As the truly min-wise independent hash functions are hard to implement, practical systems only use hash functions that approximate min-wise independence, such as functions of the MD/SHA family cryptographic hash functions.

Theorem 1. (Broder's Theorem): *For any two super-chunks S_1 and S_2, with $h(S_1)$ and $h(S_2)$ being the corresponding sets of the chunk fingerprints of the two super-chunks, respectively, where h is a hash function that is selected uniformly and at random from a min-wise independent family of cryptographic hash functions. Then (3) is established. It points out that the probability that two super-chunks have the same minimum chunk fingerprint is the same as their resemblance.*

$$\Pr(\min\{h(S_1)\} = \min\{h(S_2)\}) = r \tag{3}$$

We consider a generalization of Broder's Theorem, given in [21], for any two super-chunks S_1 and S_2, and then we have a conclusion expressed in (4) when k is far smaller than the chunk count in super-chunk, where min_k denotes the k smallest elements in a set. It means that the probability that two super-chunks share at least one fingerprint in their k smallest chunk fingerprints can increase with k. We define the k smallest chunk fingerprints of a super-chunk as its *handprint*, k is the handprint size and those chunk fingerprints in the handprint are the representative fingerprints of the super-chunk. The sampling rate of the handprinting is k over the number of chunks in the super-chunk. It is obviously that we can find more redundancy in datasets by exploiting strong ability to detect similarity with handprinting technique.

$$\begin{aligned} &\Pr(\min_k\{h(S_1)\} \cap \min_k\{h(S_2)\} \neq \varnothing) \\ &= 1 - \Pr(\min_k\{h(S_1)\} \cap \min_k\{h(S_2)\} = \varnothing) \\ &\approx 1 - (1-r)^k \end{aligned} \tag{4}$$

We define $\hat{r}$ as the estimated resemblance of the two super-chunks by handprinting in (5). The larger k value, the more accurate the estimated resemblance is likely to be. But we need to pay for a larger handprint in storage overhead. The value of chunk fingerprint in the super-chunk handprint has a hypergeometric distribution. Since the size of the handprint is usually much smaller than the chunk number of the super-chunk, we can use the binomial distribution to approximate the hypergeometric distribution. Under this assumption, the accuracy of our estimated similarity by handprinting can be given by (6), C_k^i is the number of i-combinations from a given set of k elements. For a given small error factor ε and resemblance value r, the probability that the handprint-based estimation $\hat{r}$ is within $[r\text{-}\varepsilon, r\text{+}\varepsilon]$ is proportional to the handprint size k.

$$\hat{r} \triangleq \frac{Min_k\{h(S_1)\} \cap Min_k\{h(S_2)\}}{Min_k\{h(S_1)\} \cup Min_k\{h(S_2)\}} \tag{5}$$

$$\Pr\left(|\hat{r} - r| \leq \varepsilon\right) = \sum_{k(r-\varepsilon) \leq i \leq k(r+\varepsilon)} C_k^i \cdot r^i \cdot (1-r)^{k-i} \tag{6}$$

We evaluate the effectiveness of handprinting on super-chunk resemblance detection in the first 8MB super-chunks of four pair-wise files with different application types, including Linux 2.6.7 versus 2.6.8 kernel packages, and pair-wise versions of PPT, DOC and HTML files. We actually use the Two-Threshold Two-Divisor (TTTD) chunking algorithm [25] to subdivide the super-chunk into small chunks with 1KB, 2KB, 4KB and 32KB as minimum threshold, minor mean, major mean and maximum

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/TCC.2017.2710043, IEEE Transactions on Cloud Computing

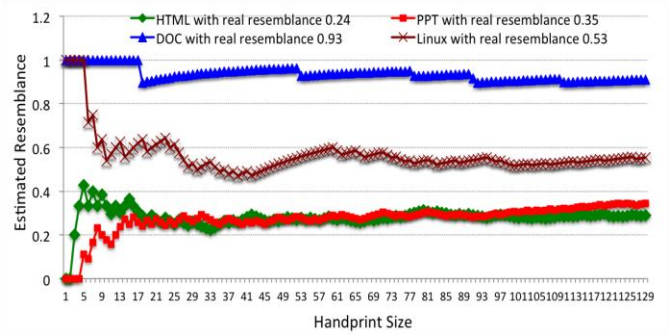


Fig. 1. The effect of handprinting resemblance detection as a function of handprint size. The real resemblance values for each applications are given in the legend entries.

threshold of chunk size, respectively. TTTD is a variant of the basic content defined chunking (CDC) algorithm [26] that leads to superior deduplication by setting absolute size limits on chunk sizes to increase average chunk size while maintaining a reasonable duplication elimination ratio. We can calculate the real resemblance values, which are shown in the legend entries, based on the Jaccard index by the whole chunk fingerprint comparison on each pair of super-chunks, and estimate the resemblance by comparing representative fingerprints in handprint comparison with different handprint sizes. The estimated resemblance, as shown in Fig. 1 as a function of the handprint size, approaches the real resemblance value as the handprint size increases. An evaluation of Fig. 1 suggests that a reasonable handprint size can be chosen in the range from 4 to 16 representative fingerprints. Comparing with the conventional schemes [6], [7], [9] that only use a single representative fingerprint (when handprint size equals to 1), our handprinting method can find more similarity for file pairs with poor similarity (with a resemblance value of less than 0.5), such as the two PPT versions and the pair of HTML versions.

3 AppDedupe Design

In this section, we use the following three design principles to govern our AppDedupe system design:

- **Throughput.** The deduplication throughput should scale with the number of nodes by parallel deduplication across the storage nodes.
- **Capacity.** Similar data should be forwarded to the same deduplication node to achieve high duplicate elimination ratio.
- **Scalability.** The distributed deduplication system should easily scale out to handle massive data volumes with balanced workload among nodes.

To achieve high deduplication throughput and good scalability with negligible capacity loss, we design a scalable inline distributed deduplication framework in this section. In what follows, we first show the architecture of our AppDedupe system. Then we present our two-tiered data routing scheme to achieve scalable performance with high deduplication efficiency. This is followed by the description of the application-aware data structures for high deduplication throughput in deduplication nodes.

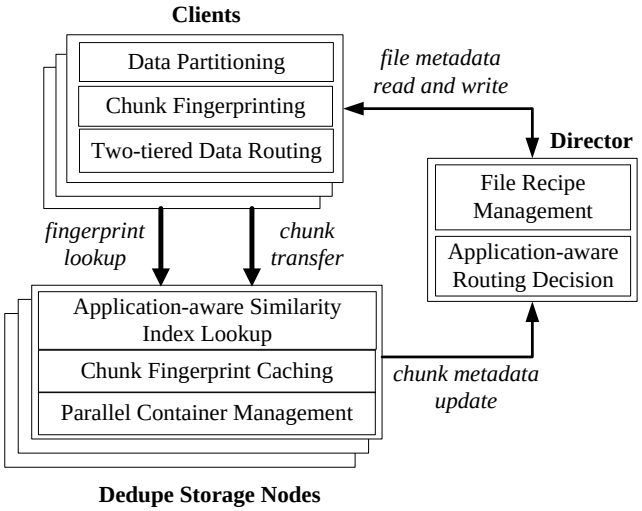


Fig.2. The architectural overview of AppDedupe.

3.1 System Overview

The architecture of our distributed deduplication system is shown in Fig. 2. It consists of three main components: clients, dedupe storage nodes and director.

Clients. There are three main functional modules in a client: data partitioning, chunk fingerprinting and data routing. The client component stores and retrieves data files, performs data chunking with fixed or variable chunk size and super-chunk grouping in the data partitioning module for each data stream, and calculates chunk fingerprints by a collision-resistant hash function, like MD5, SHA-1 or SHA-2, then routes of each super-chunk to a dedupe storage node with high similarity by the two-tiered data routing scheme. To improve distributed system scalability by saving the network transfer bandwidth during data store, the clients determine whether a chunk is duplicate or not by batching chunk fingerprint query in the deduplication node at the super-chunk level before data chunk transfer, and only the unique data chunks are transferred over the network.

Dedupe storage nodes. The dedupe server component consists of three important functional modules: application-aware similarity index lookup, chunk index cache management and parallel container management. It implements the key deduplication and storage management logic, including returning the results of application-aware similarity index lookup for data routing, buffering the recent hot chunk fingerprints in chunk index cache to speedup the process of identifying duplicate chunks and storing the unique chunks in larger units, called containers, in parallel.

Director. It is responsible for keeping track of files on the dedupe storage node, and managing file information to support data store and retrieve. It consists of file recipe management and application-aware routing decision. The file recipe management module keeps the mapping from files to chunk fingerprints and all other information required to reconstruct the file. All file-level metadata are maintained in the director. The application aware routing decision module selects a group of corresponding appli-

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/TCC.2017.2710043, IEEE Transactions on Cloud Computing

cation storage nodes for each file, and gives the client a feedback to direct super-chunk routing. The director supports up to two servers in an active/passive failover to avoid the single node failure with high availability.

3.2 Two-tiered data routing scheme

As a new contribution of this paper, we present the two-tiered data routing scheme including: the file-level application aware routing decision in director and the super-chunk level similarity aware data routing in clients.

The application aware routing decision is inspired by our application difference redundancy analysis in Section 2.2. It can distinguish from different types of application data by exploiting application awareness with filename extension, and selects a group of dedupe storage nodes as the corresponding application storage nodes, which have stored the same type of application data with the file in routing. This operation depends on an application route table structure that builds a mapping between application type and storage node ID. The application aware routing algorithm is shown in Algorithm 1, which performs in the application aware routing decision module of director.

The similarity aware data routing scheme is a stateful data routing scheme motivated by our super-chunk resemblance analysis in Section 2.3. It routes similar super-chunk to the same dedupe storage node by looking up storage status information in only one or a small number of nodes, and achieves near-global capacity load balance without high system overhead (as described in Section 4). In the data-partitioning module, a file is first divided it into c small chunks, which are grouped into a super-chunk S. Then, all the chunk fingerprints $\{fp_1, fp_2, \ldots, fp_c\}$ are calculated by a cryptographic hash function in the chunk fingerprinting module. The data routing algorithm, shown in Algorithm 2, performs in the data routing module of the clients.

Our handprinting based data routing scheme can improve load balance for the m application dedupe storage nodes by adaptively choosing least loaded node in the k candidate nodes for each super-chunk. We have proved that the global load balance can be approached by virtue of the universal distribution of randomly generated handprints by cryptographic hash functions in Section 4.5. Its consistent hashing based data assignment is scalable since it can avoid re-shuffling all previously stored data when adding or deleting a node in the storage cluster.

3.3 Interconnect communication

The interconnect communication is critical for the design of AppDedupe. We detail the operations carried out when storing and retrieving a file.

A file store request is processed as shown in the Fig. 3: a client sends a *PutFileReq* message to the director after file partitioning and chunk fingerprinting. The message includes file metadata like: file ID (the SHA-1 value of file content), file size, file name, timestamp, the number of super-chunk in the file and their checksums. The director stores the file metadata as a file recipe [34], and makes sure that there has enough space in the distributed storage systems for the file. It also performs the application

Algorithm 1. *Application Aware Routing Algorithm*

Input: the full name of a file, *fullname*, and a list of all dedupe storage nodes $\{S_1, S_2, \ldots, S_N\}$

Output: a ID list of application storage node, $ID\_list=\{A_1, A_2, \ldots, A_m\}$

1. Extract the filename extension as the application type from the file full name *fullname*, sent from client side;
2. Query the application route table in director, and find the dedupe storage node A_i that have stored the same type of application data; We get the corresponding application storage nodes $ID\_list=\{A_1, A_2, \ldots, A_m\} \subseteq \{S_1, S_2, \ldots, S_N\}$;
3. Check the node list: if $ID\_list=\varnothing$ or all nodes in *ID_list* are overloaded, then add the dedupe storage node S_L with lightest workload into the list $ID\_list=\{S_L\}$;
4. Return the result *ID_list* to the client.

Algorithm 2. *Handprinting Based Stateful Data Routing*

Input: a chunk fingerprint list of super-chunk S in a file, $\{fp_1, fp_2, \ldots, fp_c\}$, and the corresponding application storage node ID list of the file, $ID\_list=\{A_1, A_2, \ldots, A_m\}$

Output: a target node ID, i

1. Select the k smallest chunk fingerprints $\{rfp_1, rfp_2, \ldots, rfp_k\}$ as a handprint for the super-chunk S by sorting the chunk fingerprint list $\{fp_1, fp_2, \ldots, fp_c\}$, and sent the handprint to k candidate nodes with IDs mapped by consistent hashing in the m corresponding application storage nodes;
2. Obtain the count of the existing representative fingerprints of the super-chunk S in the k candidate nodes by comparing the representative fingerprints of the previously stored super-chunks in the application-aware similarity index, are denoted as $\{r_1, r_2, \ldots, r_k\}$;
3. Calculate the relative storage usage, which is a node storage usage value divided by the average storage usage value, to balance the capacity load in the k candidate nodes, are denoted as $\{w_1, w_2, \ldots, w_k\}$;
4. Choose the dedupe storage node with ID i that satisfies $r_i/w_i = \max\{r_1/w_1, r_2/w_2, \ldots, r_k/w_k\}$ as the target node.

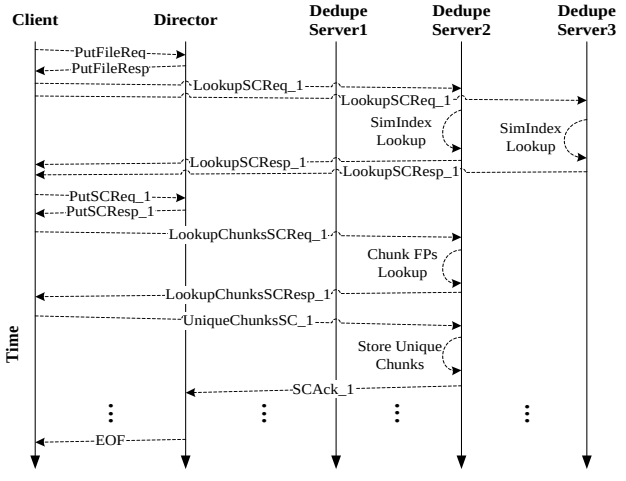


Fig.3. Message exchanges for store operation

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/TCC.2017.2710043, IEEE Transactions on Cloud Computing

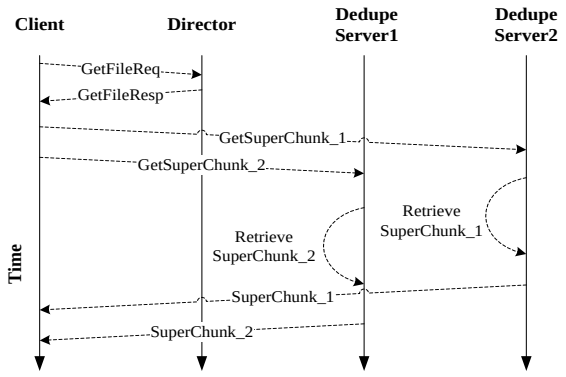


Fig.4. Message exchanges for retrieve operation

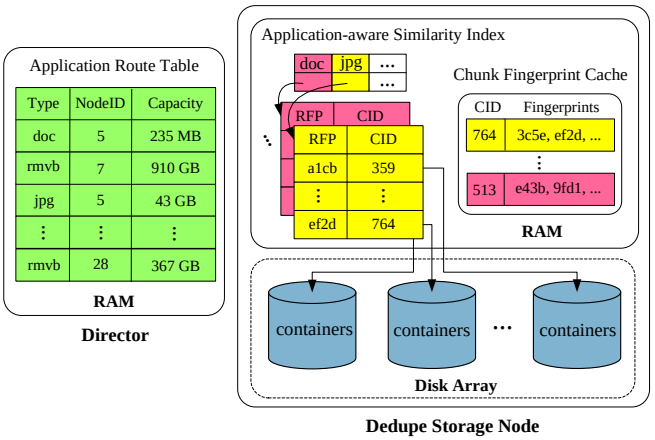


Fig.5. Key data structures in dedupe process.

aware routing decision to select a group of corresponding application storage nodes for each file. The director replies to the client with the file ID and a corresponding application storage node list in *PutFileResp* message. After received the *PutFileResp*, the client sends *k LookupSCReq* requests to the *k* candidate dedupe storage nodes for each super-chunk in the file, respectively, to lookup the application-aware similarity index in dedupe storage nodes for the representative fingerprints of the super-chunk. These candidate nodes reply to the client with a weighted resemblance value for the super-chunk. The client selects a candidate node as the target route node to store the super-chunk, and notifies the director its node ID by *PutSC Req* message. Then, the client sends all chunk fingerprints of the super-chunk in batch to the target node to identify whether a chunk is duplicated or not. After the lookup of chunk fingerprints, the target dedupe storage node replies to the client with a list of unique chunks in the super-chunk. Moreover, the client only needs to send the unique chunks in the super-chunk to the target node in batch. We repeat the steps for each super-chunk, until the end of file is reached.

The process of retrieving a file is also initiated by a client request *GetFileReq* to the director, as depicted in Fig. 4. The director reacts to this request by querying the file recipe, and forwards the *GetFileResp* message to the client. The *GetFileResp* contains the super-chunk list in the file and the mapping from super-chunk to the dedupe storage node where it is stored. Then, the client requests each super-chunk in the file from the corresponding dedupe storage node with *GetSuperChunk* message. The dedupe server can retrieve super-chunk from data containers, and the performance of restore process can be accelerated, like [35] [36]. Finally, the client downloads each super-chunk and uses the checksums of super-chunks and file ID to verify the data integrity.

3.4 Some key data structures

We outline the salient features of the key data structures designed for the deduplication process in the director and dedupe storage nodes. As shown in Fig. 5, an application route table is located in the director to conduct application aware routing decision, while to support high deduplication throughput with low system overhead, a chunk fingerprint cache and two key data structures: application-aware similarity index and container, are introduced in our dedupe storage architecture design.

Application route table is built in director to conduct application aware routing decision. Each entry of the table stores a mapping from application type to node ID and the corresponding capacity for that kind of application data in the storage node. The director can find out the application storage node list for a given application type, and calculate the workload level with storage utilization in nodes. In consideration of the application level and node level are both coarse grained, the whole application route table can easily be fit into the director memory to speedup the query operations.

Application-aware similarity index is an in-memory data structure. It consists of an application index and small hash-table based indices classified by application type. According to the accompanied file type information, the incoming super-chunk is directed to a small index with the same file type. Each entry contains a mapping between a representative fingerprint (RFP) of super-chunk handprint and the container ID (CID) where it is stored. Since our handprinting has very low sampling rate, it is much smaller than the traditional chunk fingerprint disk index that builds a mapping from all chunk fingerprints to the corresponding containers that they're stored in. To support concurrent lookup operations in application-aware similarity index by multiple data streams on multicore deduplication nodes, we adopt a parallel application-aware similarity index lookup design and control the synchronization scheme by allocating a lock per hash bucket or for a constant number of consecutive hash buckets.

Container is a self-describing data structure stored in disk to preserve locality, similar to the one described in [3], that includes a data section to store data chunks and a metadata section to store their metadata information, such as chunk fingerprint, offset and length. Our dedupe server design supports parallel container management to allocate, deallocate, read, write and reliably store containers in parallel. For parallel data store, a dedicated open container is maintained for each coming data stream, and a new one is opened up when the container fills up.

Besides the forementioned data structures, the chunk

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/TCC.2017.2710043, IEEE Transactions on Cloud Computing

fingerprint cache also plays an important role in deduplication performance improvement. It keeps the chunk fingerprints of recently accessed containers in RAM. Once a representative fingerprint is matched by a lookup request in the application-aware similarity index, all the metadata section belonging to the mapped container are prefetched into the cache to speedup chunk fingerprint lookup. When the cache is full, a reasonable cache replacement policy, like Least-Recently-Used(LRU), is applied to make room for future prefetching and caching.

4 Evaluation

We have implemented a prototype of AppDedupe in user space using C++ and pthreads, on the Linux platform. We evaluate the parallel deduplication efficiency in the multi-core deduplication server with real system implementation, while use trace-driven simulation to demonstrate how AppDedupe outperforms the state-of-the-art distributed deduplication techniques in terms of deduplication efficiency and system scalability. In addition, we conduct sensitivity studies on chunking strategy, chunk size, super-chunk size, handprint size and cluster size.

4.1 Evaluation Platform and Workload

We use four commodity servers to perform our experiments to evaluate parallel deduplication efficiency in single-node dedupe server. All of them run Ubuntu 14.10 and use a configuration with 4-core 8-thread Intel X3440 CPU running at 2.53 GHz and 16GB RAM and a Seagate ST1000DM 1TB hard disk drive. In our prototype deduplication system, 7 desktops serve as the clients, one server serves as the director and the other three servers for dedupe storage nodes. It uses Huawei S5700 Gigabit Ethernet switch for internal communication. To achieve high throughput, our client component is based on an event-driven, pipelined design, which utilizes an asynchronous RPC implementation via message passing over TCP streams. All RPC requests are batched in order to minimize the round-trip overheads. We also perform event-driven simulation on one of the four servers to evaluate the distributed deduplication techniques in terms of deduplication ratio, load distribution, memory usage and communication overhead.

We collect five kinds of real-world datasets and two types of application traces for our experiments. The Linux dataset is a collection of Linux kernel source code from versions 1.0 through 3.3.6, which is downloaded from the website [27]. The VM dataset consists of 2 consecutive monthly full backups of 8 virtual machine servers (3 for Windows and 5 for Linux). The audio, video and photo datasets are collected from personal desktops or laptops. The mail and web datasets are two traces collected from the web-server and mail server of the CS department in FIU [28]. The key workload characteristics of these datasets are summarized in Table 4. Here, the "size" column represents the original dataset capacity, and "deduplication ratio" column indicates the ratio of logical to physical size after deduplication with 4KB fixed chunk size in static chunking (SC) or average 4KB variable chunk size in optimized content defined chunking (CDC) based on the open source code in Cumulus [29].

TABLE 4
The Workload Characteristics of The Real-World Datasets and Traces

Datasets	Size (GB)	Deduplication Ratio
Linux	160	8.23(CDC) / 7.96(SC)
VM	313	4.34(CDC) / 4.11(SC)
Audio	158	1.39(CDC) / 1.21(SC)
Video	366	1.83(CDC) / 1.14(SC)
Photo	208	1.58(CDC) / 1.35(SC)
Mail	526	10.52(SC)
Web	43	1.9(SC)

4.2 Evaluation Metrics

The following evaluation metrics are used in our evaluation to comprehensively assess the performance of our prototype implementation of AppDedupe against the state-of-the-art distributed deduplication schemes.

Deduplication efficiency(DE): It is first defined in [22], to measure the efficiency of different dedupe schemes in the same platform by feeding a given dataset. It is calculated by the difference between the logical size L and the physical size P of the dataset divided by the deduplication process time T. So, deduplication efficiency can be expressed in (7).

$$DE = \frac{L - P}{T} \tag{7}$$

Normalized deduplication ratio(NDR): It is equal to the distributed deduplication ratio (DDR) divided by the single-node deduplication ratio (SDR) achieved by a single-node, exact deduplication system, and can be expressed in (8). This is an indication of how close the deduplication ratio achieved by a distributed deduplication method is to the ideal distributed deduplication ratio.

$$NDR = \frac{DDR}{SDR} \tag{8}$$

Normalized effective deduplication ratio(NEDR): It is equivalent to normalized deduplication ratio divided by the value of 1 plus the ratio of standard deviation σ of physical storage usage to average usage α in all dedupe servers, similar to the metric used in [7]. Normalized effective deduplication ratio can be expressed in (9). It indicates how effective the data routing schemes are in eliminating the deduplication node information island.

$$NEDR = \frac{DDR}{SDR} \times \frac{\alpha}{\alpha + \sigma} \tag{9}$$

Number of fingerprint index lookup messages: It includes that of inter-node messages and intra-node messages for chunk fingerprint lookup, both of which can be easily obtained in our simulation to estimate communication overhead.

RAM usage for intra-node deduplication: It is an essential system overhead related to chunk index lookup in dedupe server. And it indicates how efficient the chunk index lookup optimization is to improve the performance of intra-node deduplication.

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/TCC.2017.2710043, IEEE Transactions on Cloud Computing

Data skew for distributed storage: We define *DS* as a metric for data skew in the dedupe storage server cluster. It can be expressed in (10), and equals to the difference between the maximum capacity *MaxLoad* and the minimum capacity *MinLoad* in storage cluster divided by the mean value *MeanLoad*.

$$DS = \frac{MaxLoad - MinLoad}{MeanLoad} \quad (10)$$

4.3 Application-aware Deduplication Efficiency

In our design, the client performs data partitioning and chunk fingerprinting in parallel before data routing decision. It can divide the files into small chunks with fix-sized SC or variable-sized CDC chunking methods for each kind of application files, and calculates the chunk fingerprints with cryptographic hash function. Then, hundreds of or thousands of consecutive smaller chunks are grouped together as a super-chunk for data routing. The implementation of the hash fingerprinting is based the OpenSSL library. According to the study in [19], we select SHA-1 to reduce the probability of hash collision for fix-sized SC chunking, while we choose MD5 for variable-sized CDC chunking for high hashing throughput with almost the same hash collision possibility.

To exploit the multi-core or many-core resource of the dedupe storage node, we also develop parallel application-aware similarity index lookup in individual dedupe servers. For our multiple-data-stream based parallel deduplication, each data stream has a deduplication thread, but all data streams share a common hash-table based application-aware similarity index in each dedupe server. We lock the hash-table based application-aware similarity index by partitioning the index at the application granularity to support concurrent lookup. As we demonstrated in [33], the single-node parallel deduplication perform the best in application-aware-similarity-index lookup when the number of data streams equals to that of supported CPU logical cores, while the performance of more streams drops when the number of locks is larger than the number of data stream because of the overhead of thread context switching that causes data swapping between cache and memory.

We compare our application aware similarity index with the traditional similarity index in [33] for parallel deduplication throughput in single dedupe storage node with multiple data streams. The results in Fig. 6 show the parallel deduplication throughput using the VM dataset, with data input from RAMFS to eliminate the performance interference of the disk I/O bottleneck. We test the throughput of both traditional similarity index (Naive) and application aware similarity index (Application aware) with cold cache or warm cache, respectively. Here, "cold cache" means the chunk fingerprint cache is empty when we first perform parallel deduplication with multiple streams on the VM dataset. While "warm cache" means the duplicate chunk fingerprint had already been stored in the cache, when we perform parallel deduplication with multiple streams again on the same dataset. We observe that the parallel deduplication schemes with application-aware similarity index perform much better

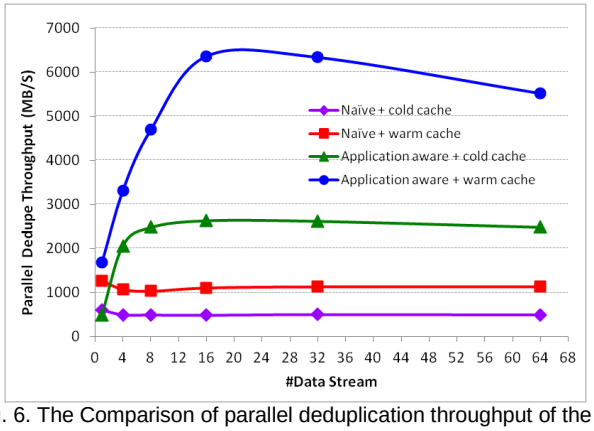


Fig. 6. The Comparison of parallel deduplication throughput of the application aware similarity index structure with that of the traditional similarity index structure for multiple data streams.

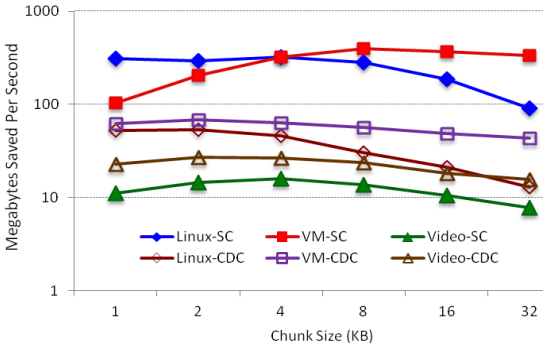


Fig.7. Deduplication efficiency in single storage node.

than the naïve parallel deduplication mechanisms, and the parallel deduplication schemes with warm cache can achieve higher throughput than those regimes with cold cache. The throughput of the parallel deduplication with both application aware similarity index and warm cache goes up to 6.2GB/s with the increasing number of data streams; After 16 concurrent streams, the throughput falls to 5.5GB/s since the concurrency overheads of index locking and disk I/O are becoming obvious.

We measure the deduplication efficiency in a configuration with two clients, one director and a single dedupe storage node to show the tradeoff between deduplication effectiveness and system overhead. To eliminate the impact of the disk bottleneck, we store the entire workload in memory and perform the deduplication process to skip the unique-data-chunk store step. To assess the impact of the system overhead on deduplication efficiency, we measure "*Bytes Saved Per Second*", the deduplication efficiency as defined in Section 4.2, as a function of the chunk size. The results in Fig. 7 show that the best choices of chunking method and chunk size to achieve high deduplication efficiency vary with different application datasets. The single dedupe server can achieve the highest deduplication efficiency when the chunk size is 4KB for statically chunked Linux workload, 8KB for statically chunked VM workload and 2KB for Video workload with CDC. As a result, we choose to perform application-aware chunking, which adaptively select the best chunking scheme and the best chunk size for each application with high deduplication efficiency.

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/TCC.2017.2710043, IEEE Transactions on Cloud Computing

TABLE 5
RAM USAGE OF INTRA-NODE DEDUPLICATION FOR VARIOUS APPLICATION WORKLOADS

Work-loads	Statless& Stateful	Extreme Binning	Produck	Σ-Dedupe	App-Dedupe
Linux	9.9MB	118.5MB	763.1MB	6.1MB	6.2MB
VM	37MB	47.2KB	2.8GB	38.6MB	23.1MB
Mail	25MB	—	1.9GB	14.3MB	15.6MB
Web	11.3MB	—	852.8MB	6.9MB	7.1MB
Audio	52.3MB	2.45MB	3.9GB	47.1MB	34.2MB
Video	162MB	103.6KB	12.6GB	101.4MB	106.8MB
Photo	74.8MB	7MB	5.6GB	53MB	49.5MB
Total	**372.3MB**	**128.1MB**	**2.84GB**	**267.4MB**	**242.5MB**

Since the disk based chunk fingerprint index is always too large to fit it into the limited RAM space, an efficient RAM data structure is needed to improve the index lookup performance and keep high deduplication accuracy. In EMC super-chunk based routing schemes, Bloom Filter [31] is employed to improve its full index lookup performance. A file-level in-RAM primary index over chunk-level on-disk indices is provided in Extreme Binning to achieve excellent RAM economy. In AppDedupe, we design an application-aware similarity index in RAM to speedup the chunk fingerprint index lookup process. To compare the RAM overhead of these three intra-node deduplication methods, we show their RAM usage on the seven different application workloads in Table 5, to eliminate more than 95% duplicate chunks, with 8KB mean chunk size in the Bloom Filter design, 48B primary index entry size for Extreme Binning, and 40B entry size for chunk index of Produck and application-aware similarity index of our design. We cannot estimate the RAM usage for Extreme Binning on the two traces (i.e., Mail and Web) without file metadata information. It is clearly that our design outperforms the traditional index lookup design of Produck, and the Bloom Filter design of EMC stateless and stateful schemes in shrinking the RAM usage for our sampling based index design. AppDedupe also outperforms the previous Σ-Dedupe design with less RAM usage due to its dynamic chunk size selection in application-aware chunking scheme. Our application-aware similarity index occupies more space than file-level primary index in Extreme Binning for datasets with large files, like VM and multimedia workloads, but less for Linux source code dataset with small files. We can further reduce its size by adjusting super-chunk size or handprint size with the corresponding deduplication effectiveness loss.

4.4 Load Balance in Super-Chunk Assignment

Load balance is an important issue in distributed storage technique [41]. It can help improve system scalability by en-suring that client I/O requests are distributed equitably across a group of storage servers. The implementation of consistent hashing based DHTs in traditional distributed deduplication [6], [9], [10] are considerable load imbalance due to its stateless assignment design. In particular,

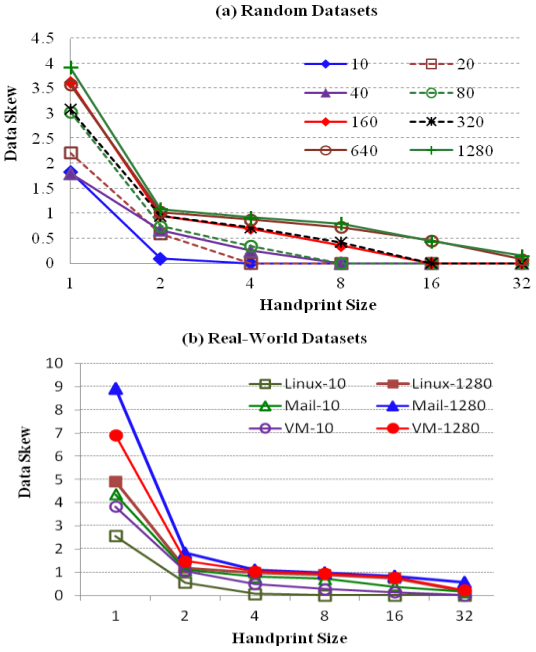


Fig. 8. The data skew of super-chunk routing when we select different handprint sizes under various node numbers in the storage server cluster.

a storage node that happens to be responsible for a larger segment of the keyspace will tend to be assigned a greater number of data objects. In subsection 4.2, we defined *DS* as a metric for data skew in the storage server cluster. Data assignment method can achieve global load balance when *DS*=0. The larger *DS* value we measured in the test, the more serious load imbalance happened in the storage cluster.

To make our conclusion more general, we not only consider an ideal scenario that each super-chunk is filled with random data in Fig. 8(a), but also perform the tests on real-world datasets: VM images, Linux source code, and mail datasets with 10 and 1280 nodes, respectively, as shown in Fig. 8(b). Super-chunk chooses the least loaded node in the k storage nodes that are mapped independently and uniformly at random by consistent hashing with the k representative fingerprints in its handprint. The effectiveness of load balancing has been tested with different handprint size and node number in Fig. 8. The average load of each node is 65536 super-chunks with 1 MB size. The important implication of the results is that even a small amount of representative fingerprints in handprint can lead to drastically different results in load balancing. It shows that traditional schemes, like DHT [9] or hash-bucket [10], with only one representative fingerprint for choice are hard to keep a good load balance in the super-chunk assignment. When the cluster scales to hundreds or thousands of nodes, our handprint technique can keep a good load balance (the metric of data skew is less than 1) by routing the super-chunks with $k \geq 4$ random choices. We select handprint size from 4 to 16 to make a tradeoff between load balance and communication overhead for large-scale distributed deduplication in big data.

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/TCC.2017.2710043, IEEE Transactions on Cloud Computing

4.5 Distributed Deduplication Efficiency

We route data at the super-chunk granularity to preserve data locality for high performance of distributed deduplication, while performing deduplication at the chunk granularity to achieve high deduplication ratio in each server locally. Since the size of the super-chunk is very sensitive to the tradeoff between the index lookup performance and the distributed deduplication effectiveness, as demonstrated by the sensitivity analysis on super-chunk size in [7], we also choose the super-chunk size of 1MB to reasonably balance the conflicting objectives of cluster-wide system performance and capacity saving, and to fairly compare our design with the previous EMC distributed deduplication mechanism.

In this section, we first conduct a sensitivity study to select an appropriate handprint size for our AppDedupe scheme, and then compare our scheme with the state-of-the-art approaches that are most relevant to AppDedupe, including EMC's super-chunk based Stateful and Stateless data routing, Σ-Dedupe, Produck and Extreme Binning, in terms of the effective deduplication ratio, normalized to that of the traditional single-node exact deduplication, and communication overhead measured in number of fingerprint index lookup messages. We emulate each node by a series of independent fingerprint lookup data structures, and all results are generated by trace-driven simulations on the seven datasets under study.

Two-tiered data routing can accurately direct similar data to the same dedupe server by exploiting application awareness and data similarity. We conduct a series of experiments to demonstrate the effectiveness of distributed deduplication by our handprint-based deduplication technique with the super-chunk size of 1MB on the Linux kernel source code workload. Fig. 9 shows the deduplication ratio, normalized to that of the single-node exact deduplication, as a function of the handprint size. As a result, AppDedupe becomes an approximate deduplication scheme whose deduplication effectiveness nevertheless improves with the handprint size because of the increased ability to detect similarity in super chunks with a larger handprint size (recall Section 2.3). We can see that there is a significant improvement in normalized deduplication ratio for all cluster sizes when handprint size is larger than 8. This means that, for a large percentage of super-chunk queries, we are able to find the super-chunk that has the largest content overlap with the given super-chunk to be routed by our handprint-based routing scheme. To strike a sensible balance between the distributed deduplication ratio and system overhead, and match the handprint size choice in single-node, we choose a handprint consisting of 8 representative fingerprints in the following experiments to direct data routing on super-chunks of 1MB in size.

After the selection on super-chunk size and handprint size, we conduct prelimary experiments on AppDedupe prototype in small scale to compare it with the two distributed deduplication schemes with application-aware-routing-only (AppAware) and hand-print-based-stateful-routing-only (∑-Dedupe), respectively. We feed the distributed dedupe storage system with the chunk fingerprints of the above described seven application workloads totalled 1774 GB size, to eliminate the disk I/O bottleneck. In Table 6, in addition to the defined metrics in Subsection 4.2, we also define *application distribution* as the number of application type distributed in three dedupe storage nodes, record *time* spent for the deduplication processes and indicate *capacity* to describe the total storage capacity after distributed deduplication in the storage cluster. The results in key metrics are shown that our AppDedupe performs best in deduplication efficiency with high deduplication ratio and low time overhead. AppAware scheme has the highest deduplication ratio, but it suffers from load imbalance due to the data skew of application level data assignment. ∑-Dedupe suffers from a low deduplication ratio since it distributes all types of application data into each dedupe storage node with the highest cross-node redundancy.

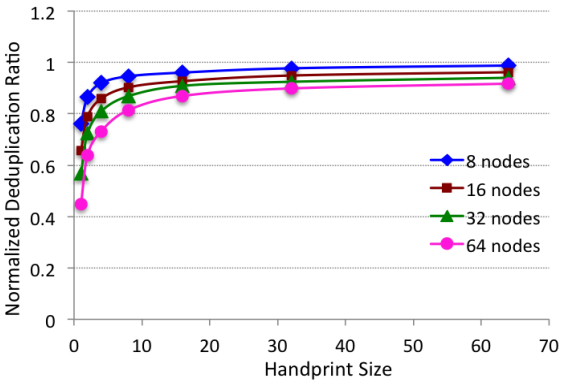


Fig. 9. Distributed deduplication ratio normalized to that of single-node exact deduplication with various cluster size, as a function of handprint size.

TABLE 6
KEY METRICS OF THE DISTRIBUTED DEDUPLICATION SCHEMES

Method	Application Distribution	Capacity (GB)	DS	EDR	Time(s)	DE (MB/s)
AppAware	2:2:3	610	1.84	0.45	28335	41.8
Σ-Dedupe	7:7:7	657	0.05	0.87	30250	36.6
AppDedupe	3:4:4	618	0.07	0.95	23775	48.6

To show the scalability of our AppDedupe design, we also perform simulations to compare it with the existing data routing schemes in distributed deduplication in large scale. We compare our AppDedupe scheme with the state-of-the-art data routing schemes of Extreme Binning, Produck, Σ-Dedupe, EMC's stateless routing scheme and stateful routing scheme, across a range of datasets. Fig. 10 plots EDR as a function of the cluster size for the six algorithms on all datasets. Because the last two traces, Mail and Web, do not contain file-level information, we are not able to perform the file-level based Extreme Binning scheme on them. In general, App-Dedupe can achieve the highest effective deduplication ratio due to its load balancing design of inter-node two-tiered data routing and high deduplication effectiveness of intra-node application-aware chunking by leveraging application awareness and data similarity. More specifically, the AppDedupe scheme achieves 103.1% of the EDR obtained by the State-

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/TCC.2017.2710043, IEEE Transactions on Cloud Computing

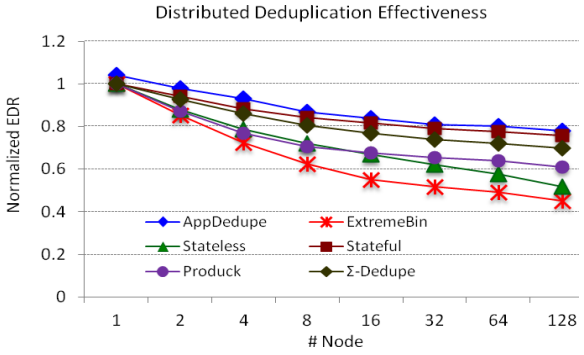


Fig.10. Effective deduplication ratio (EDR), normalized to that of single-node exact deduplication, as a function of cluster size on workloads.

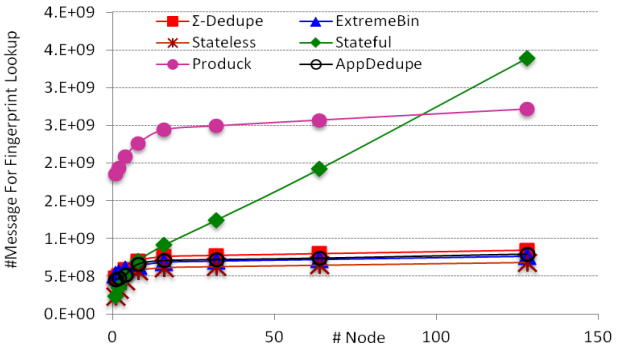


Fig.11. Communication overhead in terms of the number of fingerprint-lookup message for all seven workloads.

ful scheme for a cluster of 128 server nodes on the seven datasets, while this performance margin narrows to 103.6% when averaging over all cluster sizes, from 1 through 128. Stateless routing performs worse than AppDedupe, Produck and Stateful routing due to its low cluster-wide data reduction ratio and unbalanced capacity distribution. Extreme Binning underperforms Stateless routing on the workloads because of the large file size and skewed file size distribution in the datasets, workload properties that tend to render Extreme Binning's similarity detection ineffective. Produck achieves higher normalized EDR than Stateless routing and Extreme Binning due to its stateful routing design, but it underperforms AppDedupe on all datasets for its low deduplication ratio and unbalanced load distribution. AppDedupe outperforms Extreme Binning in EDR by up to 72.7% for a cluster of 128 nodes on the five datasets containing file-level information. For the seven datasets, AppDedupe is better than Stateless routing in EDR by up to 50.5% for a cluster of 128 nodes. Our AppDedupe achieves an improvement of 28.2% and 11.8% in EDR with respect to Produck and Σ-Dedupe on all datasets in a cluster of 128 nodes, respectively. As can be seen from the trend of curves, these improvements will likely be more pronounced with cluster sizes larger than 128.

In distributed deduplication storage systems, fingerprint lookup tends to be a persistent bottleneck in each dedupe storage server because of the costly on-disk lookup I/Os, which often adversely impacts the system scalability due to the consequent high communication overhead from fingerprint lookup. To quantify this system overhead, we adopt the number of fingerprint-lookup messages as a metric. We measure this metric by totaling the number of chunk fingerprint-lookup messages on the seven datasets, for the five distributed deduplication schemes. As shown in Fig. 11 that plots the total number of fingerprint-lookup messages as a function of the node number, AppDedupe, Extreme Binning and Stateless routing have very low system overhead due to their constant fingerprint-lookup message count in the distributed deduplication process, while the number of fingerprint-lookup messages of Stateful routing grows linearly with the node number. This is because Extreme Binning and Stateless routing only have 1-to-1 client-and-server fingerprint lookup communications for source deduplication due to their stateless designs. Stateful routing, on the other hand, must send the fingerprint lookup requests to all nodes, resulting in 1-to-all communication that causes the system overhead to grow linearly with the node number even though it can reduce the overhead in each node by using a sampling scheme. Produck has high communication overhead due to its fine-grained chunk size with 1KB, while other deduplication methods adapt 4KB or 8KB. The number of fingerprint-lookup messages in Produck is about four times that of AppDedupe, Extreme Binning and Stateless routing, and it grows as slow as these three low-overhead schemes. As described in Algorithm 1, the main reason for the low system overhead in AppDedupe is that the pre-routing fingerprint-lookup requests for each super-chunk only need to be sent to at most 8 candidate nodes, and only for the lookup of representative fingerprints, which is 1/32 of the number of chunk fingerprints, in these candidate nodes. The message overhead of AppDedupe in fingerprint lookup is about 1.25 times that of Stateless routing and Extreme Binning in all scales. Σ-Dedupe is the preliminary version of our AppDedupe, and they have almost the same communication overhead due to their consistent interconnect protocol.

5 Conclusion

In this paper, we describe AppDedupe, an application-aware scalable inline distributed deduplication framework for big data management, which achieves a tradeoff between scalable performance and distributed deduplication effectiveness by exploiting application awareness, data similarity and locality. It adopts a two-tiered data routing scheme to route data at the super-chunk granularity to reduce cross-node data redundancy with good load balance and low communication overhead, and employs application-aware similarity index based optimization to improve deduplication efficiency in each node with very low RAM usage. Our real-world trace-driven evaluation clearly demonstrates AppDedupe's significant advantages over the state-of-the-art distributed deduplication schemes for large clusters in the following important two ways. First, it outperforms the extremely costly and poorly scalable stateful tight coupling scheme in the cluster-wide deduplication ratio but only at a slightly higher sys-

tem overhead than the highly scalable loose coupling schemes. Second, it significantly improves the stateless loose coupling schemes in the cluster-wide effective deduplication ratio while retaining the latter's high system scalability with low overhead.

ACKNOWLEDGMENT

This research was partially supported by the National Key Research and Development Program of China under Grant 2016YFB1000302, the National Natural Science Foundation of China under Grant 61232003 and 61402518, and the US NSF under Grants CNS-1116606 and CNS-1016609. A preliminary version of the paper was presented at the 2012 ACM/IFIP/USENIX Middleware Conference.

REFERENCES

[1] J. Gantz, D. Reinsel, "The Digital Universe Decade-Are You Ready?" White Paper, IDC, May 2010.

[2] H. Biggar, "Experiencing Data De-Duplication: Improving Efficiency and Reducing Capacity Requirements," White Paper, the Enterprise Strategy Group, Feb. 2007.

[3] K.R. Jayaram, C. Peng, Z. Zhang, M. Kim, H. Chen, H. Lei. "An Empirical Analysis of Similarity in Virtual Machine Images," Proc. Of the ACM/IFIP/USENIX Middleware Industry Track Workshop (Middleware'11), Dec. 2011.

[4] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti. "iDedup: Latency-aware, inline data deduplication for primary storage," Proc. of the 10th USENIX Conference on File and Storage Technologies (FAST'12). Feb. 2012.

[5] P. Shilane, M. Huang, G. Wallace, and W. Hsu. "WAN optimized replication of backup datasets using stream-informed delta compression," ACM Transactions on Storage (TOS), 8(4): 915-921, Nov. 2012.

[6] D. Bhagwat, K. Eshghi, D.D. Long, M. Lillibridge, "Extreme Binning: Scalable, Parallel Deduplication for Chunk-based File Backup," Proc. of the 17th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'09), pp.1-9, Sep. 2009.

[7] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, P. Shilane, "Tradeoffs in Scalable Data Routing for Deduplication Clusters," Proc. of the 9th USENIX Conf. on File and Storage Technologies (FAST'11), pp. 15-29, Feb. 2011.

[8] F. Douglis, D. Bhardwaj, H. Qian, P. Shilane, "Content-aware Load Balancing for Distributed Backup," Proc. of the 25th USENIX Conf. on Large Installation System Administration (LISA'11), pp.151-168, Dec. 2012.

[9] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepko-wski, C. Ungureanu, M. Welnicki, "HYDRAstor: a Scalable Secondary Storage," Proc. of the 7th USENIX Conf. on File and Storage Technologies (FAST'09), pp. 197-210, Feb. 2009.

[10] J. Wei, H. Jiang, K. Zhou, D. Feng, "MAD2: A Scalable High Throughput Exact Deduplication Approach for Network Backup Services," Proc. of the 26th IEEE Conf. on Mass Storage Systems and Technologies (MSST'10), pp. 1-14, May 2010.

[11] T. Yang, H. Jiang, D. Feng, Z. Niu, K. Zhou, Y. Wan, "DEBAR: a Scalable High-Performance Deduplication Storage System for Backup and Archiving," Proc. of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS'10), pp. 1-12, Apr. 2010.

[12] H. Kaiser, D. Meister, A. Brinkmann, S. Effert, "Design of an Exact Data Deduplication Cluster," Proc. of the 28th IEEE Symposium on Mass Storage Systems and Technologies (MSST'12), pp. 1-12, Apr. 2012.

[13] B. Zhu, K. Li, H. Patterson, "Avoiding the Disk Bottleneck in the Data Domain Deduplication File System," Proc. of the 6th USENIX Conf. on File and Storage Technologies (FAST'08), pp. 269-282, Feb. 2008.

[14] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deola- likar, G. Trezise, P. Camble, "Sparse indexing: large scale, inline deduplication using sampling and locality," Proc. of the 7th Conf. on File and Storage Technologies (FAST '09), pages 111-123, Feb. 2009.

[15] B. Debnath, S. Sengupta, and J. Li. "ChunkStash: speeding up inline storage deduplication using flash memory." In Proceedings of the 2010 USENIX conference on USENIX annual technical conference (2010), USENIX Association, p. 16.

[16] W. Xia, H. Jiang, D. Feng, Y. Hua, "Silo: a Similarity-locality based Near-exact Deduplication Scheme with Low RAM Overhead and High Throughput," Proc. of 2011 USENIX Annual Technical Conference (ATC'11), pp. 285-298, Jun. 2011.

[17] A. Katiyar, J. Weissman. "ViDeDup: An Application-Aware Framework for Video Deduplication," Proc. of the 3rd USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'11), pp. 31-35, 2011.

[18] C. Liu, Y. Lu, C. Shi, G. Lu, D. Du, and D.-S Wang, "ADMAD: Application-driven metadata aware de-deduplication archival storage systems," Proc. of the 5th IEEE International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI'08), pp.29-35, 2008.

[19] Y. Fu, H. Jiang, N. Xiao, L. Tian, F. Liu, L. Xu. "Application-Aware Local-Global Source Deduplication based Cloud Backup Services for Personal Storage." In Proc. of IEEE Transactions on Parallel and Distributed Systems, 25(5): 1155-1165, 2014.

[20] L. Aronovich, R. Asher, E. Bachmat, H. Bitner, M. Hirsch, and S. T. Klein, "The design of a similarity based deduplication system," Proc. of 2nd ACM Annual Int. Systems and Storage Conf. (SYSTOR '09), pp. 6-6, Jun. 2009.

[21] D. Bhagwat, K. Eshghi, P. Mehra, "Content-based Document Routing and Index Partitioning for Scalable Similarity-based Searches in a Large Corpus," Proc. of the 13th ACM International Conf. on Knowledge Discovery and Data Mining (SIGKDD'07), pp. 105-112, Aug. 2007.

[22] Y. Fu, H. Jiang, N. Xiao, L. Tian, F. Liu, "AA-Dedupe: An Application-Aware Source Deduplication Approach for Cloud Backup Services in the Personal Computing Environment," Proc. of the 13th IEEE Conf. on Cluster Computing (Cluster'11), pp. 112-120, Sep. 2011.

[23] Jaccard Index. http://en.wikipedia.org/wiki/Jaccard_index. 2012.

[24] A.Z. Broder, M. Charikar, A.M. Frieze, M. Mitzenmacher, "Min-wise Independent Permutations," Journal of Computer and System Sciences, vol. 60, no. 3, pp. 630–659, Jun. 2000, doi:10.1006/jcss.1999.1690.

[25] K. Eshghi, H.K. Tang, "A framework for Analyzing and Improving Content-based Chunking Algorithms," Technical Report, HPL-2005-30R1, HP Lab., Palo Alto, Sep. 2005.

[26] A. Muthitacharoen, B. Chen, D. Mazieres, "A low-bandwidth network file system," Proc. of the 18th ACM symposium on Operating systems principles (SOSP'01), pp.174-187, Oct. 2001.

[27] The Linux Kernel Archives. http://www.kernel.org/, 2012

[28] FIU IODedup Traces. http://iotta.snia.org/traces/391, 2010

[29] M. Vrable, S. Savage, G.M. Voelker, "Cumulus: Filesystem Backup to the Cloud," Proc. of the 8th USENIX Conf. on File and Storage Technologies (FAST'09), pp. 225-238, Feb. 2009.

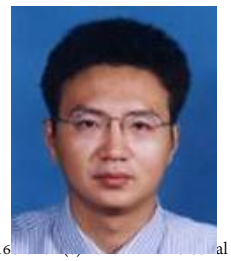
[30] P. Efstathopoulos, "File Routing Middleware for Cloud Deduplication," Proc. of 2nd ACM Interanational Workshop on Cloud Computing Platforms (CloudCP'12), Apr. 2012.

[31] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," Communications of the ACM, vol. 13, no. 7, pp. 422-426, Jul. 1970.

[32] D. Frey, A-M. Kermarrec, K. Kloudas, "Probabilistic Deduplication for Cluster-Based Storage Systems," Proc. of the 3rd ACM Symposium on Cloud Computing (SOCC'12), pp. 1-

19, Oct. 2012.
[33] Y. Fu, H. Jiang, N. Xiao, "A Scalable Inline Cluster Deduplication Framework for Big Data Protection," Proc. of the 13th ACM/IFIP/ USENIX Conf. on Middleware (Middleware'12), pp. 354-373, Dec. 2012.
[34] D. Meister, A. Brinkmann, T. Sub, "File Recipe Compression in Data Deduplication Systems," Proc. of the 11th USNIX Conf. on File and Storage Technologies (FAST'13), Feb. 2013.
[35] M. Lillibridge, K. Eshghi, D. Bhagwat, "Improving Restore Speed for Backup Systems that Use Inline Chunk-Based Deduplication," Proc. of the 11th USNIX Conf. on File and Storage Technologies (FAST'13), Feb. 2013.
[36] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, W. Xia, F. Huang, Q. Liu. "Accelerating Restore and Garbage Collection in Deduplication-based Backup Systems via Exploiting Historical Information," Proc. Of the USENIX Annual Techinical Conference (ATC'14), Jun. 2014.
[37] Q. Liu, Y. Fu, G. Ni, R. Hou. "Hadoop Based Scalable Cluster Deduplication for Big Data." Proc. Of the ICDCS Workshops 2016: 98-105.
[38] S. Dewakar, S. Subbiah, G. Soundararajan, M. Wilson, M.W. Storer, K. Udayashankar, K. Voruganti, M. Shao, "Storage Efficiency Opportunities and Analysis for Video Repositories," In 7th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'15), May 2015.
[39] A. Shimi, R. Kalach, A. Kumar, J. Li, A. Oltean, and S. Sengupta, "Primary Data Deduplication-Large Scale Study and System Design," Proc. of the USENIX Annual Techinical Conference (ATC'12), Jun. 2012.
[40] Min Fu, Dan Feng, Yu Hua, Xubin He, Zuoning Chen, Wen Xia, Yucheng Zhang, Yujuan Tan. "Design Tradeoffs for Data Deduplication Performance in Backup Workloads," Proc. of the 13th USNIX Conf. on File and Storage Technologies (FAST'13), pp. 331-344, Feb. 2015.
[41] Min Xu, Yunfeng Zhu, Patrick P. C. Lee, Yinlong Xu. "Even Data Placement for Load Balance in Reliable Distributed Deduplication Storage Systems," In Proc. of IEEE International Symposium on Quality of Service (IWQoS), pp. 349-358, 2015.
[42] Yan-Kit Li, Min Xu, Chun-Ho Ng, and Patrick P. C. Lee. "Efficient Hybrid Inline and Out-of-line Deduplication for Backup Storage," ACM Transactions on Storage, 11(1): 1-21, 2014.
[43] Joao Paulo and Jose Pereira. "Efficient Deduplication in a Distributed Primary Storage Infrastructure," ACM Transactions on Storage, 12(4): 1-35, 2016.

Yinjin Fu is an assistant professor in computer science in PLA University of Science and Technology. He received the B.S. degree in Mathematics from Nanjing University, Nanjing, China, in 2006, the M.S. degree and Ph.D. degree in computer science from the College of Computer at National University of Defense Technology, Changsha, China, in 2008 and 2013, respectively. He joined the Department of Computer Science and Engineering at University of Nebraska-Lincoln as a visiting scholar from 2010 to 2012. His current research interests include data deduplication, cloud storage and distributed file systems. He has more than 30 publications in journals and international conferences including IEEE TPDS, ACM ToS, Commun. Lett., JCST, MIDDLEWARE, MSST, CLUSTER, FCS, ICA3PP, NAS. He is a member of the IEEE, ACM and CCF.



Nong Xiao received the B.S. and Ph.D. degrees in computer science from the College of Computer at National University of Defense Technology (NUDT) in China, in 1990 and 1996, respectively. He is currently a professor in the State Key Laboratory of High Performance Computing at NUDT, China. His current research interests include large-scale storage system, network computing, and computer architecture. He has more than 130 publications to his credit in journals and international conferences including IEEE TPDS, TSC, TC, TMM, JPDC, JCST, HPCA, ICCAD, MIDDLEWARE, MSST, DATE, DAC, IPDPS, CLUSTER, CLOUD, SYSTOR and MASCOTS. He is a distinguished member of the CCF and a member of the IEEE and ACM.

Hong Jiang received the B.S. degree in computer engineering from Huazhong University of Science and Technology, Wuhan, China, in 1982, the M.S. degree in computer engineering from the University of Toronto, Canada, in 1987, and the Ph.D. degree in computer science from the Texas A&M University, College Station, in 1991. Since August 1991, he has been at the University of Nebraska-Lincoln (UNL), where he served as the vice chair of the Department of Computer Science and Engineering (CSE), and is a professor of CSE. Now he is the chair of Department of Computer Science and Engineering at University of Texas at Arlington. His present research interests include computer architecture, computer storage systems and parallel I/O, parallel/distributed computing, cluster and grid computing. He has more than 180 publications in major journals and international conferences in these areas, including IEEE TPDS, IEEE TC, JPDC, USENIX-ATC, ISCA, MICRO, FAST, ICDCS, IPDPS, OOPLAS, ECOOP, SC, ICS, MIDDLEWARE, HPDC, ICPP, etc. He is a fellow of the IEEE and a member of the ACM and ACM SIGARCH.

Guyu Hu received the B.S. degree in radiotechnics from Univeristy of Zhejiang, Hangzhou, China, in 1983, the Master degree and Ph.D. degree in communication engineering from PLA Institution of Communication Engineering, Nanjing, China, in 1989 and 1992, respectively. Now he is a professor in network engineering of PLA University of Science and Technology. His current research interests include computer network and machine learning. He has more than 160 publications in journals and conferences in these areas. He is a senior member of the Chinese Institute of Electronics.

Weiwei Chen received her B.S. degree in computer science from PLA Institute of Electronic Technology, Zhengzhou, China, in 1990, the Master degree in computer science from National University of Defense Technology, Changsha, China. Now she is a professor in computer science of PLA University of Science and Technology. Her current research interests include computer storage and cloud computing. She has more than 30 publications in journals and conferences in these areas, and is a member of the IEEE and China Computer Federation.