Similarity and Locality Based Indexing for High Performance Data Deduplication

Wen Xia, Hong Jiang, *Senior Member, IEEE*, Dan Feng, *Member, IEEE*, and Yu Hua, *Senior Member, IEEE*

Abstract—Data deduplication has gained increasing attention and popularity as a space-efficient approach in backup storage systems. One of the main challenges for centralized data deduplication is the scalability of fingerprint-index search. In this paper, we propose SiLo, a near-exact and scalable deduplication system that effectively and complementarily exploits similarity and locality of data streams to achieve high duplicate elimination, throughput, and well balanced load at extremely low RAM overhead. The main idea behind SiLo is to expose and exploit more similarity by grouping strongly correlated small files into a segment and segmenting large files, and to leverage the locality in the data stream by grouping contiguous segments into blocks to capture similar and duplicate data missed by the probabilistic similarity detection. SiLo also employs a locality based stateless routing algorithm to parallelize and distribute data blocks to multiple backup nodes. By judiciously enhancing similarity through the exploitation of locality and vice versa, SiLo is able to significantly reduce RAM usage for index-lookup, achieve the near-exact efficiency of duplicate elimination, maintain a high deduplication throughput, and obtain load balance among backup nodes.

Index Terms—Data deduplication, storage system, index structure, performance evaluation

1 Introduction

Due to the explosive growth of the digital data, data deduplication has gained increasing attention for its space efficiency in backup storage systems. Data deduplication not only reduces the storage space requirements by eliminating redundant data [1], [2], [3], [4], [5], [6], [7] but also minimizes the network transmission of duplicate data in the network storage systems [8]. It splits files into multiple chunks that are each uniquely identified by a hash signature (e.g., MD5, SHA-1, and SHA-256), also called a fingerprint [1], [4]. It removes duplicate chunks by checking their fingerprints, which avoids byte-by-byte comparisons.

Despite recent progress in data deduplication studies [4], [9], [10], many challenges remain, particularly in the petabyte-scale deduplication based backup storage systems that are generally centralized. One of the main challenges is the scalability of fingerprint-index based search schemes [4]. For example, to backup a unique data set of 1 PB and assuming an average chunk size of 8 KB, at least 2.5 TB of SHA-1 fingerprints will be generated, which are too large to be stored in the memory. State-of-the-art deduplication systems [1], [4], [9] suggest that the access throughput to the on-disk fingerprint-index is about 1-6 MB/sec, which is too low to be acceptable for backup services. Thus fingerprinting indexing has become the main performance bottleneck of large-scale data deduplication systems.

In order to address this performance bottleneck, many approaches have been proposed to improve the performance of deduplication indexing, by putting the hot fingerprints into RAM to minimize accesses to on-disk index and improve the throughput of deduplication. There are two primary approaches to scaling data deduplication: locality based acceleration of deduplication, and similarity based deduplication.

Locality-based approaches exploit the inherent locality in a backup stream, which is widely used in state-of-the-art deduplication systems such as DDFS [4], Sparse Indexing [9], and ChunkStash [11]. The locality in this context means that the chunks of a backup stream will appear in approximately the same order in each full backup with a high probability. Mining this locality increases the RAM utilization and reduces the accesses to on-disk index, thus alleviating the disk bottleneck.

Similarity-based approaches are designed to address the problem encountered by locality-based approaches in backup streams that either lack or have very weak locality (e.g., incremental backups). They exploit data similarity instead of locality in a backup stream, and reduce the RAM usage by extracting similar characteristics from the backup stream. A well-known similarity-based approach is Extreme Binning [10] that improves deduplication scalability by exploiting the file similarity to achieve a single on-disk index access for chunk lookup per file.

While these scaling approaches have significantly alleviated the disk bottleneck in data deduplication, there are still substantial limitations that prevent them from reaching the peta- or exa-scale, as explained below. Based on our analysis of experimental results, we find that in general a locality-based deduplication approach performs very poorly when the backup stream lacks locality while a

- *W. Xia, D. Feng, and Y. Hua are with the Wuhan National Laboratory for Optoelectronics, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074 China. E-mail: {xia, dfeng, csyhua}@hust.edu.cn.*
- *H. Jiang is with the Department of Computer Science and Engineering, University of Nebraska-Lincoln, 217 Schorr Center, 1101 T Street, Lincoln, NE 68588-0150. E-mail: jiang@cse.unl.edu.*

Manuscript received 21 May 2013; revised 18 Nov. 2013; accepted 24 Jan. 2014. Date of publication 24 Feb. 2014; date of current version 13 Mar. 2015. Recommended for acceptance by R.J. Figueiredo.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2014.2308181

similarity-based approach underperforms for a backup stream with a weak similarity. Unfortunately, the backup data in practice are quite complicated in how or whether locality/similarity is exhibited.

In fact, DDFS is shown to run very slowly in backup streams when there is little or no locality (e.g., users only do the incremental backup) [10], [11]. On the other hand, the similarity-based Extreme Binning approach is shown to fail to find significant amounts of duplicate data in data sets that have little or no file similarity (e.g., the files are edited frequently) [6], [12]. Fortunately, our preliminary study indicates that the judicious exploitation of locality can compensate for the lack of similarity in data sets, and vice versa. In other words, both locality and similarity can be complementary to each other, and can be jointly exploited to improve the overall performance of deduplication.

Inspired by the state-of-the-art work on exploitation of locality [4], [11] and similarity [10] for data deduplication indexing, we present SiLo, a scalable and low-overhead near-exact deduplication system, to overcome the aforementioned shortcomings of these state-of-the-art schemes. The main idea behind SiLo is to expose and exploit more similarity by grouping strongly correlated small files into a segment and segmenting large files, and to leverage the locality in the data stream by grouping contiguous segments into blocks to capture similar and duplicate data missed by the probabilistic similarity detection. SiLo also employs a locality based stateless routing algorithm to parallelize and distribute data blocks to multiple backup nodes. The main contributions of this paper include:

- SiLo proposes a new similarity algorithm that groups many small correlated files into a segment and segments large files to expose more similarity and reduce the RAM usage for index-lookup.
- SiLo mines the locality characteristics of data streams by grouping multiple contiguous segments into blocks to capture more similar and duplicate data missed by the probabilistic similarity detection and caching the recently accessed blocks in RAM to avoid frequent accesses to on-disk index.
- The combined and complementary exploitation of these two backup-stream properties overcomes the shortcomings of existing approaches based on either property alone. A mathematical model analytically evaluating this approach for deduplication indexing is developed and shown to be consistent with our experimental evaluation.
- SiLo employs a locality based stateless routing algorithm to distribute data blocks to multiple backup nodes for parallel processing, according to the representative fingerprints of data blocks while preserving the data access locality on each backup node.
- Results from experimental evaluation show that SiLo significantly improves the overall performance of deduplication by its judicious and joint exploitation of similarity & locality and outperforms two existing state-of-the-art approaches, the similarity-based "Extreme Binning" and the locality-based "ChunkStash," under various workloads. Storage load of SiLo is also shown to be well balanced while using four backup nodes in our evaluation.

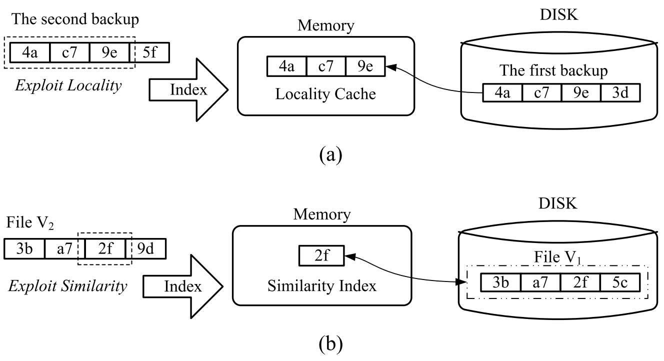


Fig. 1. Locality based and similarity based deduplication approaches. (a) Locality approach. (b) Similarity approach.

The rest of the paper is organized as follow. Background and motivation for this research are presented in Section 2. The architecture and design of SiLo are described in Section 3. Our experimental evaluation of SiLo and its comparisons with the state-of-the-art ChunkStash and Extreme Binning systems are discussed in Section 4. We summarize the related work in Section 5, draw conclusions and outline future work in Section 6.

2 Background and Motivation

In this section, we first provide the necessary backgrounds for SiLo, and then motivate our work by analyzing the observations based on extensive experiments on locality- and similarity-based deduplication accelerating approaches under real-world workloads.

2.1 Deduplication Accelerating Approaches

Chunk-based deduplication is the most widely used data reduction approach for secondary storage systems. Such a system breaks a file into contiguous chunks and eliminates duplicate chunks by identifying their secure hash digests (e.g., SHA-1) [1], [8].

As the size of data sets to be deduplicated increases, so does the total size of fingerprints required to detect duplicate chunks, which can quickly overflow the RAM capacity for even high TB-scale and low PB-scale data sets. This can result in frequent disk accesses for fingerprint-index lookups, thus severely limiting the throughput of deduplication systems. Plenty of studies [4], [9], [10], [11] have paid attention to this challenge of data deduplication which lies in the on-disk index-lookup bottleneck. Currently, there are two general approaches to accelerating the index-lookup of deduplication and alleviating the disk bottleneck, namely, the locality based and the similarity based methods.

Locality based approaches. Locality in the context of data deduplication refers to the observation that similar or identical files, say, A, B, and C (thus their data chunks), in a backup stream appear in approximately the same order throughout multiple full backups with a very high probability [4], [13]. As shown in Fig. 1a, DDFS [4], a well-known deduplication system, makes full use of this locality property by storing the chunks in the order of the first backup stream (e.g., chunks' fingerprints {4a, c7, 9e, 3d}) on the disk. Upon the lookup of fingerprint "4a" of the second backup, DDFS will prefetch the fingerprints {4a, c7, 9e} and preserve this locality in the RAM, which helps reduce the accesses to the on-disk index when looking up the fingerprints of "c7," "9e" later. It also uses Bloom filters to quickly identify new (non-duplicate) chunks, which helps compensate for the cases where there is no or little locality.

Sparse Indexing [9] improves this method by sampling index instead of using Bloom filters in face of data sets with little or no locality, which reduces more than half of the RAM usage for indexing than DDFS. As an improved chunking algorithm, Bimodal Chunking [13] proposes that the neighboring data of duplicate chunks should be assumed to be good deduplication candidates for further re-chunking due to the backup-stream locality, which helps maximize the duplicate detection.

Similarity based approaches. The similarity here refers to the similarity characteristics of a file or a data stream, for example, the maximal or minimal value of the sets of chunk fingerprints, that can be extracted to represent the file or the data stream [10]. Fig. 1b shows an example of file similarity, where two sets of chunk fingerprints, {3b, a7, 2f, 9d} and {3b, a7, 2f, 5c}, belong to files V_1 and V_2 respectively. Here the file similarity is represented by the minimal fingerprint in the hash set of a file whose prefix bits represent the smallest value among the same prefix bits of all the fingerprints there. Thus in the event of the minimal fingerprint "2f" of file V_2 being detected to be identical to that of file V_1, we can consider the two files similar and then detect duplicate chunks between file V_1 and V_2, which avoids globally indexing for the chunk fingerprints of file V_2.

Generally, the similarity-based approaches are proposed to exploit the similar characteristics of backup streams to minimize the chunk-lookup index in the memory. Extreme Binning [10] exploits this similarity among files instead of locality, allowing it to make only one disk access for chunk lookup per file. Thus it significantly reduces the RAM usage by storing only the similarity-based index in the memory. But it puts similar files in a bin whose size grows with the size of the data, resulting in decreased throughput as the size of the similarity bin increases and failing to exploit the inherent locality of backup streams [6], [12].

2.2 Deduplication of Large and Small Files

Our experimental observations, as well as intuition, suggest that the deduplication of large files can be very important while the deduplication of small files can be very time and RAM-space consuming.

Large files. A typical file system contains many large files (e.g., $\geq$2 MB) that only account for less than 20 percent of total number of files but occupy more than 80 percent of the total space [7], [14], such as VMware images and database files. A recent study also suggests that the files larger than 1 GB account more than 90 percent of the total space in backup storage systems [15], because of backup software that tends to group individual files into "tar-like" collections. Obviously, these large files are an important consideration for a deduplication system due to their high space-capacity and bandwidth/time requirements in the inline backup process. The larger the files, the less similar they will appear to be even if significant parts within the files may be similar or identical, which can cause the similarity-based approaches to miss the identification of significant redundant data in large files.

To address this problem of large files, SiLo approach divides a large file into many small segments to better expose similarity among large files. More specifically, the probability that large files S_1 and S_2 share the same representative fingerprint is highly dependent on their similarity degree according to Broder's theorem [16], [17]: *Consider two sets S_1 and S_2, with $H(S_1)$ and $H(S_2)$ being the corresponding sets of the hashes of the elements of S_1 and S_2 respectively, where H is chosen uniformly and randomly from a min-wise independent family of permutations. Let min(S) denotes the smallest element of the set of integers S. Then*

$$Pr[min(H(S_1)) = min(H(S_2))] = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}. \tag{1}$$

This probability can be increased by segmenting the large files and detecting the similarity of all the segments of the large files, as follows:

$$\begin{aligned} Pr[min(H(S_1)) = min(H(S_2))] &= \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} \\ &\ll \mathsf{Pr[min(H(S_{11})) = min(H(S_{21})) \cup \cdots \cup min(H(S_{1n}))} \\ &= \mathsf{min(H(S_{2n}))]} = \bigcup_{i=1}^{n} Pr[min(H(S_{1i})) = min(H(S_{2i}))] \\ &= 1 - \bigcap_{i=1}^{n} Pr[min(H(S_{1i})) \neq min(H(S_{2i}))] \\ &= 1 - \prod_{i=1}^{n}\left(1 - \frac{|S_{1i} \cap S_{2i}|}{|S_{1i} \cup S_{2i}|}\right). \end{aligned} \tag{2}$$

As files S_1 and S_2 are segmented into segments $S_{11}{\sim}S_{1n}$ and $S_{21}{\sim}S_{2n}$ respectively, similarity detection between S_1 and S_2 is determined by the union of the probabilities of similarity detection between $S_{11}{\sim}S_{1n}$ and $S_{21}{\sim}S_{2n}$. Based on the above probability analysis, this segmenting approach will only fail in the worst-case scenario where all the segments in file S_1 are not similar to segments of file S_2. This, based on the inherent locality in the backup streams, happens with a very small probability because it is extremely unlikely that two files are very similar but none of their respective segments is detected as being similar.

Small files. A file system typically contains a very large number of small files [7], [14]. Since the small files (e.g., $\leq$64 KB) usually only take up less than 20 percent of the total space of a file system but account for more than 80 percent of the total number of files, the chunk-lookup index for small files will be disproportionably large and likely out of memory. Consequently, the inline deduplication [4], [9] of small files will tend to be very slow and inefficient. This problem of small files can be addressed by grouping many highly correlated small files into a segment. We consider the logically adjacent files within the same parent directory to be highly correlated and thus similar. We exploit similarity and locality of a group (i.e., segment) of adjacent small files rather than one individual file or chunk.

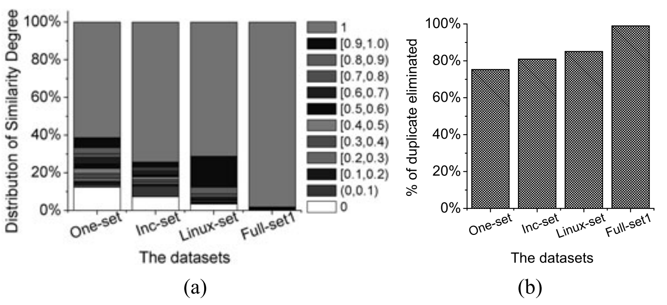


Fig. 2. Our experimental observations of the similarity-only approach on four data sets. (a) The distribution of segment similarity degree. (b) Duplicate eliminated by the similarity-only approach.

As a result, at most one access to on-disk index is needed per segment instead of per file or per chunk.

2.3 Similarity and Locality

In this section, we further analyze the relationship between similarity and locality with respect to backup streams. As our motivation, we jointly exploit similarity and locality of backup streams by grouping several chunks into a segment and grouping several segments into a block respectively. Detailed definitions of segment and block in SiLo are given as follows:

- *A segment* consists of several sequential chunks so as to expose and exploit the backup-stream similarity, whose implementation is presented in Section 3.2. We only place the similarity index (e.g., the minimal chunk-fingerprint) of each segment in the RAM to reduce the RAM usage of fingerprint indexing.
- *A block* is composed of several sequential segments so as to preserve and exploit the backup-stream locality, whose implementation is described in Section 3.3. We fully exploit the locality of recently accessed blocks by keeping their associated fingerprints in the RAM to minimize the accesses to on-disk index.

As mentioned in Section 2.2, we pack small files and segment large files into segments to exploit similarity for duplicate detection. As shown in Fig. 2, we examine the distribution of similarity degree and the duplicate elimination measure of this similarity-only deduplication approach on four data sets. The four data sets represent the workloads of the first backups, incremental backups, Linux source code files and several full-backups respectively, whose characteristics will be detailed in Section 4.1. The *similarity degree* is computed in our similarity approach by detecting the duplicates among an input segment and the currently stored segments as: Simi(S_{input} = Max) ($|S_{input} \cap S_i|$)/$|S_{input}|$, ($S_i \in S_{store}$, Simi(S_{input}) $\in$ [0, 1]). Thus, a similarity degree of "1" signifies that the input segment is completely identical to some of the currently stored segments and a similarity degree of "0" states that the segment is detected to match no other segments at all by the similarity-only approach.

Fig. 2a shows that a large percentage of segments are of low similarity degree in our four real-world data sets. Fig. 2b demonstrates that similarity-only based deduplication approach fails to remove a large proportion of duplicate data when the data sets have a large amount of duplicate data with low similarity degree (e.g., One-set and Inc-set). Therefore, similarity-based deduplication efficiency heavily depends on the similarity degree of the backup stream which is well consistent with Broder's Theorem (see Section 2.2).

In addition, Extreme Binning groups many similar files to a bin [10], which cannot meet the requirements of increasing number of backup versions. Now, introducing a group of new similar sets $S_1 {\sim} S_n$. According to aforementioned Broder's Theorem [16], all of these similar files are grouped into the same bin can be computed as follows:

$$\begin{aligned} &Pr[min(H(S_1)) = min(H(S_2)) = \cdots = min(H(S_n))] \\ &= \frac{|S_1 \cap S_2 \cap \cdots \cap S_n|}{|S_1 \cup S_2 \cup \cdots \cap S_n|} = \frac{|\bigcap_{i=1}^{n} S_i|}{|\bigcup_{i=1}^{n} S_i|} \le \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}. \end{aligned} \tag{3}$$

As a result, it can be speculated that the increasing similar files will be not put into the same bin, resulting in more duplicate data will be missed to be detected by the similarity approach [10]. Inspired by Bimodal Chunking [13], which shows that the backup stream locality can be mined to find more potentially duplicate data, we believe that such locality can also be mined to expose and thus detect more data similarity, a point well demonstrated by our experimental study in Section 4. More specifically, SiLo mines locality in conjunction with similarity by grouping multiple contiguous segments in a backup stream into a block.

Therefore, SiLo could detect potentially duplicate data chunks missed by the probabilistic similarity detection by jointly exploiting similarity and locality of the segments and blocks. For example, segments S_{11}, S_{12}, and S_{13} are similar to segments S_{21}, S_{22}, and S_{23} respectively, but only S_{11} and S_{21} have the same representative fingerprint. Then we exploit locality by grouping $S_{11} {\sim} S_{13}$ and $S_{21} {\sim} S_{23}$ into blocks B_1 and B_2. Since we detect S_{11} and S_{21} to be similar by their representative fingerprints and then load block B_1 into cache, S_{22} and S_{23} will be detected to be potentially similar to S_{12} and S_{13} respectively in the cache. Thus, this exploitation of locality helps expose more similarity and then find more potential deduplication candidates by detecting similar segments' adjacent segments.

Now we analyze deduplication efficiency of the combined exploitation of similarity and locality. Given two blocks B_1 and B_2, each containing n segments ($S_{11} {\sim} S_{1n}$, $S_{21} {\sim} S_{2n}$), according to the Broder's theorem, the percentage of duplicate eliminated by the similarity-only approach can be computed as: $Dedup_{Simi}(B_1,\ B_2 =)|B_1 \cap B_2|/|B_1 \cup B_2|$. The combined and complementary exploitation of similarity and locality can be computed as follows:

$$\begin{aligned} &DeDup_{SiLo}(B_1, B_2) \\ &= Pr\left[\bigcup_{i=1}^{n} min(H(S_{1i})) = min(H(S_{2i}))\right] \\ &= 1 - Pr\left[\bigcap_{i=1}^{n} min(H(S_{1i})) \neq min(H(S_{2i}))\right] \\ &= 1 - \prod_{i=1}^{n}\left(1 - \frac{|S_{1i} \cap S_{2i}|}{|S_{1i} \cup S_{2i}|}\right) \\ &= 1 - (1-a)^N \left(assume\ all\ the\ \frac{|S_{1i} \cap S_{2i}|}{|S_{1i} \cup S_{2i}|} = a\right). \end{aligned} \tag{4}$$

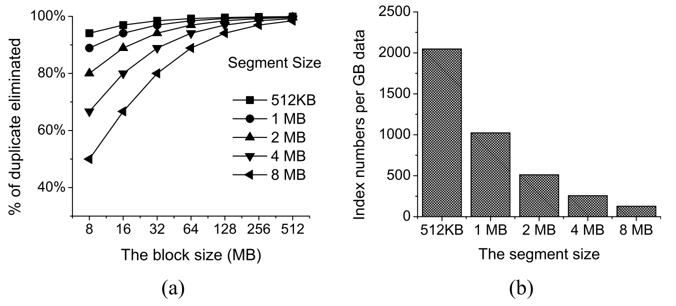


Fig. 3. Predictive analytic of combined exploitation of similarity and locality as a function of segment size and block size. (a) The percentage of duplicate data eliminated. (b) Similarity index number per GB unique data.

Assume that the value a follows a standard uniform distribution in the range [0, 1] (It may be much more complicated in the real world data sets as shown Fig. 2a), the expected value of duplicate elimination can be further calculated under the above assumption as:

$$\begin{aligned} E_{Simi} &= \int_0^1 (a)da = \frac{1}{2} \\ E_{SiLo} &= \int_0^1 (1-(1-a)^N)da = \frac{N}{N+1} \\ &= \frac{BlockSize/SegSize}{BlockSize/SegSize+1} = \frac{BlockSize}{BlockSize+SegSize}. \end{aligned} \tag{5}$$

Thus the larger the value N (i.e., the number of segments in a block), the more locality can be exploited in deduplication. E_{Simi} is equal to E_{SiLo} when N = 1. SiLo removes more than 99 percent of duplicate data when N $>$ 99. Then SiLo could achieve the performance of near-exact duplicate elimination with proper ratio of "BlockSize/SegSize" as shown in Fig. 3a while it only maintains hundreds of similarity indices in the RAM (i.e., segment size >1MB) to deduplicate a GB data with millions of chunks as depicted in Fig. 3b.

Therefore, SiLo, through its judicious and joint exploitation of locality and similarity, is able to achieve the near-exact duplicate elimination (recall that exact deduplication achieves complete duplicate elimination) and requires at most one disk access per segment (i.e., a group of contiguous chunks or small files).

3 Design and Implementation

In this section, we will first describe the architecture overview of SiLo. Then we give detailed description of its design and implementation algorithms. At the end of this section, we discuss the overall workflow of SiLo.

3.1 System Architecture Overview

SiLo is designed for large-scale and disk-inline backup storage systems. As depicted in Fig. 4, the SiLo architecture consists of four key functional components, namely, file daemon (FD), deduplication server (DS), storage server (SS), and backup server (BS), which are distributed in the data centers to serve the backup requests. BS and DS reside in the metadata server (MDS) while FD is installed on each client machine that requires backup/restore services.

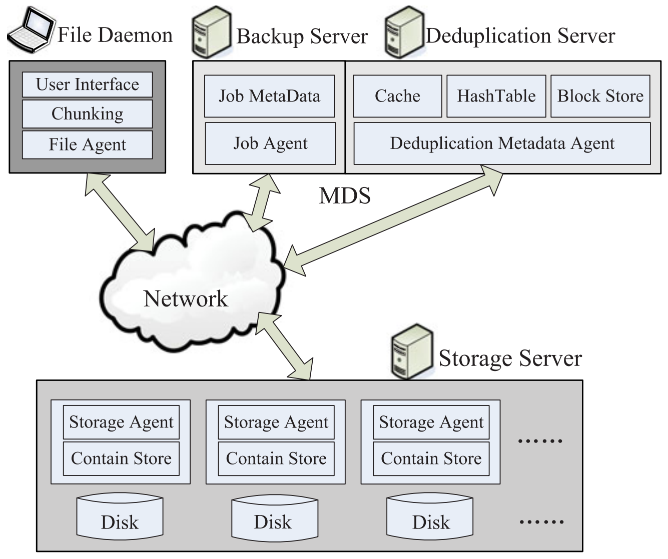


Fig. 4. The SiLo system architecture.

- *File Daemon* is a daemon program providing a functional interface (e.g., backup/restore) in users' computers. It is responsible for collecting backup data sets and sending/restoring them to/from Storage Servers for backups/restores. The processes of chunking, fingerprinting, and segmenting can be done by FD in the preliminary phase of the inline deduplication. It also includes a file agent that is responsible for communicating with BS and DS and transferring backup data to/from SS.
- *Backup Server* is the manager of the backup system that globally manages all jobs of backup/restore and directs all File Agents and Storage Servers. It maintains a metadata database for administering all backup files' information.
- *Deduplication Server* is to store and look up all fingerprints of files and chunks. It is also responsible for distributing the data blocks to store to multiple backup nodes with a stateless routing algorithm.
- *Storage Server* is the repository for backed-up data. SS in SiLo manages multiple Storage Nodes for scalability and provides fast, reliable, and safe backup/restore services.

In this paper, we focus on deduplication server since it is the most likely performance bottleneck of the entire deduplication system. DS consists of locality hash table (LHTable), similarity hash table (SHTable), write buffer, and read cache. While SHTable and LHTable index segments and blocks, the similarity and locality units of SiLo respectively, the write buffer and read cache preserve the similarity and locality of backup streams, as shown in Fig. 5. As mentioned in Section 2.3, the notion of segment is used to exploit the similarity of backup stream while the block preserves the stream-informed locality layout of segments on the disk. SHTable provides the similarity detection for input segments and LHTable serves to quickly

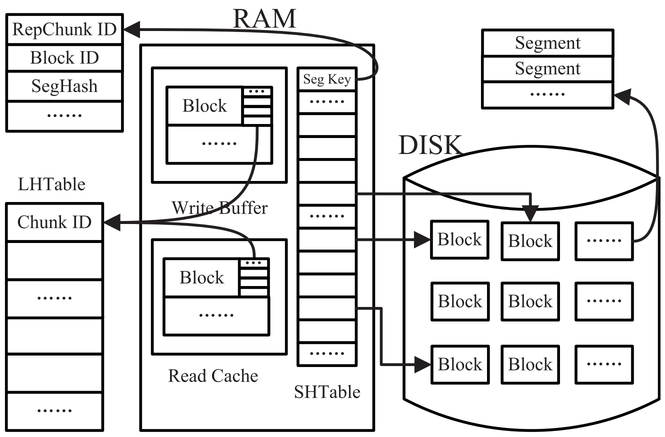


Fig. 5. Data structures of the deduplication server. RepChunkID is the representative fingerprint of a segment and Chunk ID is the SHA-1 fingerprint of a chunk.

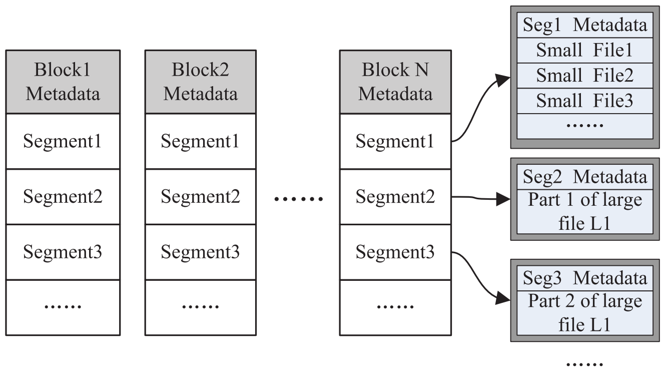


Fig. 6. Data structures of the SiLo similarity algorithm.

index and filter out duplicate chunks. The write buffer and read cache contain the recently accessed blocks to exploit the backup stream locality [4].

For the input data stream, SiLo will first use its similarity algorithm (Section 3.2) to pack correlated small files and divide large files into segments and check with SHTable to detect similarity. Then SiLo will use its locality algorithm (Section 3.3) to enhance the similarity detection thus find more duplicate data. Finally, SiLo will distribute the data-block to different storage nodes by a locality based stateless routing algorithm (Section 3.4).

Note that, since this paper mainly aims to improve the indexing performance by making highly efficient use of the cache space of RAM and reducing accesses to on-disk fingerprints in the deduplication system, all write/read operations of segments/blocks in this paper are performed in the form of writing/reading chunks' fingerprints while operations on the data-block are performed on the real backup data.

3.2 Similarity Algorithm

As mentioned in Section 2.3, SiLo improves deduplication index scalability by combined exploitation of similarity and locality. It exploits similarity by grouping strongly correlated small files and segmenting large files, while locality is exploited by grouping contiguous segments in a backup stream to preserve the locality layout of these segments as depicted in Fig. 6. Thus, segments are the atomic building units of a block that is in turn the atomic unit of write buffer and read cache and blocks are indexed by the unique block ID over the lifetime of the system.

As a key contribution of SiLo, the SiLo similarity algorithm is implemented in file daemon, which structures data from backup streams into segments after the processing of chunking and fingerprinting according to the following three principles.

- P1. Fingerprint set of correlated small files in a backup stream (e.g., those under the same parent directory) are to be grouped into a segment.
- P2. Fingerprint set of a large file in a backup stream is divided into several independent segments.
- P3. All segments are of approximately the same size (e.g., 2 MB).

Where, P1 aims to reduce the RAM overhead of index-lookup; P2 helps expose more similarity characteristics of large files to eliminate more duplicate data; and P3 simplifies the management of segments. Thus, the similarity algorithm exposes and then exploits more similarity by leveraging file semantics and preserving locality-layout of a backup stream to significantly reduce the RAM usage for deduplication indexing.

The method of representative fingerprinting [10] is employed in SiLo to represent each segment by a similarity-index entry in the similarity hash table. By virtue of P1, the SiLo similarity design solves the problem of small files taking up disproportionably large RAM space. For example, assuming an average segment size of 2 MB and an average chunk or small file size of 8 KB, a segment accommodates 250 chunks or small files, thus significantly reducing the required index size in the memory. If we assume a 60-byte primary key for the similarity indexing of a 4 MB segment (backup data), which is considered economic, a 1 PB backup stream only needs 15 GB similarity-index for deduplication that can easily fit in the memory. Therefore, SiLo is able to use a very small and proper portion of RAM to support PB-scale deduplication by virtue of its similarity algorithm.

3.3 Locality Approach

As another salient feature of SiLo, the SiLo locality algorithm groups several contiguous segments in a backup stream into a block and preserves their locality-layout on the disk. The methods and workflow of this locality algorithm are depicted in Fig. 7. According to the locality characteristic of backup streams, if input segment S_{1i} in block B_1 is determined to be similar to segment S_{2k} by hitting in similarity hash table, SiLo will consider the whole block B_1 to be similar to block B_2 that contains S_{2k}. As a result, this grouping of contiguous segments into a block can eliminate more potentially duplicate data that is missed by the probabilistic similarity detection, thus complementing the similarity detection.

Since the block is the minimal write/read unit of write buffer and read cache in the SiLo system, it serves to maximize the RAM utilization and reduce frequent accesses to on-disk index by retaining access locality of the backup stream. When SiLo reads the blocks from disk by the similarity detection, it puts the recently accessed block into

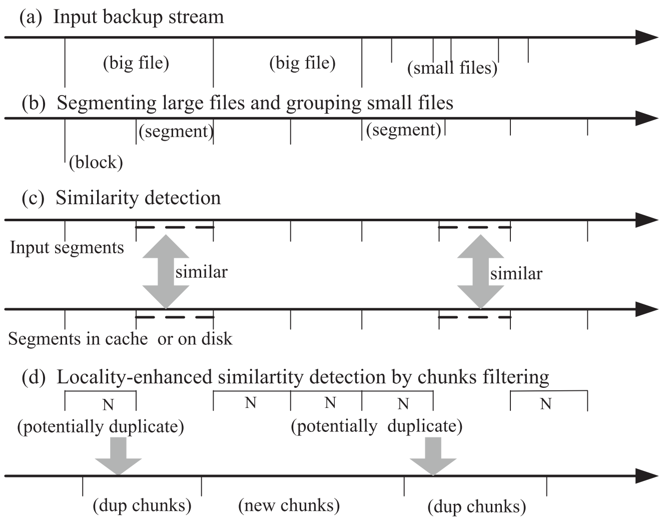


Fig. 7. The workflow of the locality algorithm: it helps detect more potentially duplicate chunks that are missed by the similarity detection. "N" refers to the fact that the segment is detected as dissimilar.

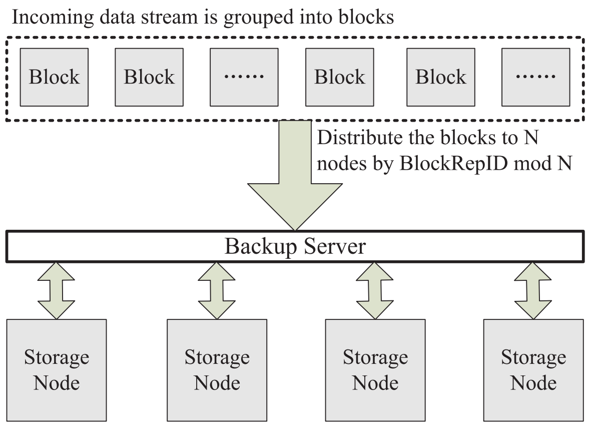

Incoming data stream is grouped into blocks
Block
Block
......
Block
Block
......
Distribute the blocks to N nodes by BlockRepID mod N
Backup Server
Storage Node
Storage Node
Storage Node
Storage Node

Fig. 8. The locality-based routing algorithm of SiLo.

the read cache. By preserving the backup-stream locality in the read cache, the accesses to on-disk index due to similarity detection can be significantly reduced, which alleviates the disk bottleneck and increases the deduplication throughput.

The block size is an important system parameter that affects the system performance in terms of duplicate elimination and throughput (see Fig. 3). The smaller the block size, the more disk accesses will be required by the server to read the index, weakening the locality exploitation. The larger the block size, on the other hand, the more unrelated segments will be read by the server from the disk, increasing system's space and time overheads for deduplication indexing. Therefore, a proper block size not only provides good duplicate elimination, but also achieves high throughput and low RAM usage in the SiLo system.

Each block in SiLo has its own locality hash table (i.e., LHTable shown in Fig. 5) for chunk filtering. Since a block contains several segments, it needs an indexing tool for thousands of fingerprints. The fingerprints in a block are organized into the LHTable when reading the block from the disk. The additional time required for constructing LHTable in a block is significantly compensated by its quick indexing.

Beside the locality of read cache, we also exploit the locality of write buffer for data deduplication. Since users of file systems tend to duplicate files or directories under the same directories, a significant amount of duplicate data can be also eliminated by detecting the duplication in write buffer that also preserves the locality of a backup stream. For example, a code directory may include many versions of source code files or documents that become good deduplication candidates.

In our current design of SiLo, the read cache and write buffer each contains a fixed number of blocks. Only a very small portion of RAM is thus used as the write buffer and read cache to store a small number of recently accessed blocks to avoid the frequent and expensive disk read/write operations. As illustrated in Figs. 5 and 6, a locality-block contains only metadata information such as LHTable, segment information, chunk information, and file information, which enables a 1 MB locality-block to represent a 200 MB data-block.

3.4 Load Distribution

To meet the requirement of increasing size of data sets and scale up the deduplication throughput, the system should distribute and parallelize backup streams to multiple storage nodes. Inspired by the previous stateless routing approaches [10], [12], SiLo proposes a locality-based stateless routing algorithm that also makes full use of the aforementioned similarity and locality of backup streams. Specifically, the representative fingerprints [10] of blocks (i.e., BlockRepID) are examined and updated when the input segments are grouped into blocks to determine which backup node a data-block should be stored, as shown in Fig. 8. For example, if there are N backup nodes, the block with representative fingerprint F_i will be allocated to backup node $F_i \; mod \; N$.

This locality based routing technique is designed to maximize the scalability and availability of the deduplication based data storage while maintaining the exploitation of similarity and locality for deduplication indexing. Moreover, the data on a backup node may eventually be split into two approximately equal parts when more nodes are added into the system or when this node becomes overly loaded due to either disproportionally high access popularity or amount of data stored. The block with representative fingerprint F_i on the original backup node $F_i \; mod \; N$ will be rebalanced into the new node $F_i \; mod \; (N+1)$, which avoids rebalancing the data globally in the deduplication system.

The benefits of this locality-based distributed approach of SiLo are twofold. First, SiLo's process of routing a data-block to a backup node is stateless, thus independent of any knowledge or content of backup nodes. Due to the randomness property of hash digest algorithms (e.g., SHA-1), the representative fingerprint of a locality-aware block is a suitable choice for distributing the blocks among multiple storage nodes and obtaining load balance. Second, SiLo also preserves the backup stream locality in each backup node while minimizing the data dependencies among backup nodes. Thus, the data is stored in a backup node instead of being fragmented across multiple nodes. This means that

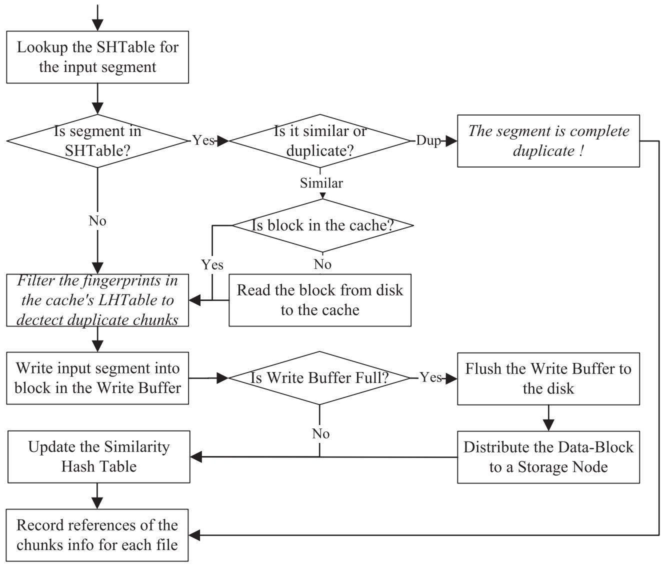

Lookup the SHTable for the input segment
Is segment in SHTable?
Yes
Is it similar or duplicate?
Dup
The segment is complete duplicate !
Similar
No
Is block in the cache?
Yes
No
Filter the fingerprints in the cache's LHTable to dectect duplicate chunks
Read the block from disk to the cache
Write input segment into block in the Write Buffer
Is Write Buffer Full?
Yes
Flush the Write Buffer to the disk
No
Update the Similarity Hash Table
Distribute the Data-Block to a Storage Node
Record references of the chunks info for each file

Fig. 9. The workflow of SiLo deduplication.

the access locality is preserved in a node so that tasks such as data restore, data deletion, and garbage collection do not have to frequently chase dependencies spanning multiple backup nodes.

3.5 SiLo Workflow

To put things together and in perspective, Fig. 9 shows the main workflow of SiLo deduplication processes. Files in the backup stream are first chunked, fingerprinted, and packed into segments by grouping strongly correlated small files and segmenting large files in the file agent. For an input segment S_{new}, SiLo goes through the following key steps:

- Check to see if S_{new} is in the SHTable. If it hits in SHTable, SiLo checks if the block B_{bk} containing S_{new}'s similar segment is in the cache. If it is not in the cache, SiLo will load B_{bk} from the disk to the read cache according to the referenced block ID of S_{new}'s similar segment, where a block is replaced in the FIFO order if the cache is full.
- The duplicate chunks in S_{new} are detected and eliminated by checking the fingerprint sets of S_{new} with LHTable (fingerprints index) of B_{bk} in the cache.
- If S_{new} misses in SHTable, it is then checked against recently accessed blocks in the read cache for potentially similar segment (i.e., locality-enhanced similarity detection).

Then SiLo will construct input segments into blocks to retain access locality of the input backup stream. For an input block B_{new}, SiLo does following:

- The representative fingerprint of B_{new} will be examined to determine the stored backup nodes of data-block B_{new}.
- SiLo checks if the write buffer is full. If the write buffer is full, a block there is replaced in the FIFO order by B_{new} and then written to the disk.

After the process of deduplication indexing, SiLo will record the chunk-to-file mapping information as the reference for each file, which is managed by the job metadata of the backup server. For the read operation, SiLo will read the referenced metadata of each target file in the job metadata that allows the corresponding data chunks to be read from the data blocks in the storage server. These data chunks will then be used to reconstruct the target files in the file daemon according to the index mapping relationship between files and deduplicated data chunks.

4 Performance Evaluations

In this section, we first describe the experimental setup of performance evaluations on SiLo protype implementation. Then, we examine several important design parameters of SiLo system to provide useful insights, and compare SiLo with two state-of-the-art approaches Extreme Binning and ChunkStash in the key deduplication metrics of duplicate elimination, RAM usage, and throughput. At the end of this section, we discuss experimental results of SiLo's load distribution algorithm.

4.1 Experimental Setup

We conduct our performance evaluation of SiLo on a platform of standard server configuration to evaluate and compare the inline deduplication performances of SiLo, ChunkStash, and Extreme Binning approaches running on a Linux environment. The hardware configuration includes a quad-core CPU running at 2.4 GHz, with a 4 GB RAM, 2 gigabit network interface cards, and two 500 GB 7,200 rpm hard disks.

Due to our lack of access to the source code of either the ChunkStash or Extreme Binning scheme, we have chosen to implement both of them. More specifically, we have implemented the locality-based and exact-deduplication approach of ChunkStash incorporating the principles and algorithms described in the ChunkStash paper [11]. The ChunkStash approach makes full use of the inherent locality of backup streams and uses a novel data structure called Cuckoo hash for fingerprint indexing. We have also implemented a simple version of the Extreme Binning approach, which represents a similarity-based and approximate-deduplication approach according to the algorithms described in the Extreme Binning paper [10]. Extreme Binning exploits file similarity instead of locality in the backup streams.

Note that our evaluation platform is not a production-quality deduplication system but rather a research prototype. Hence, our evaluation results should be interpreted as an approximate and comparative assessment of the three systems above, and not be used for absolute comparisons with other deduplication systems. The RAM usage in our evaluation is obtained by recording the memory allocated for index-lookup. The duplicate elimination performance metric is defined as the percentage of duplicate data eliminated by the system. Throughput of the system is measured by the rate at which fingerprints of the backup stream are processed, not the real backup throughput in that it does not measure the rate at which the backup data is transferred and stored.

Five traces representing different strengths of locality and similarity are used in the performance evaluation of the three deduplication systems and are listed in Table 1. The

TABLE 1
Workload Characteristics of the Five Traces Used in the Performance Evaluation

Feature	One-set	Inc-set	Linux	Full-set1	Full-set2
Total Size	530GB	251GB	101GB	2.51TB	6.0TB
Total files	3.5M	0.59M	8.8M	11.3M	12.1M
Total chunks	51.7M	29.4M	16.9M	417.6M	1.87G
Average chunk size	10KB	8KB	5.9KB	6.5KB	4.9KB
Dedup factor	1.7	2.7	19	25	16.7
Locality	weak	weak	strong	strong	strong
Similarity	weak	strong	strong	strong	strong

All use SHA-1 for chunk fingerprints and the content-defined Chunking algorithm. The deduplication factor is defined as the totalsize/(totalsize-dedupsize) ratio.

five traces are collected from real-world data sets of One-backup, Incremental-backup, Linux-version, and two Full-backup sets respectively.

- One-set trace was collected from 15 graduate students of our research group. Since we obtain only one full backup for this group, this trace has weak locality and weak similarity.
- Inc-set is a subset of the trace reported by Tan et al. [18] and was collected from initial full backups and subsequent incremental backups of eight members in a research group. There are 391 backups with a total of 251 GB data. Since the operation of incremental backup only backs up the modified and new files after the first full backup, Inc-set represents data sets with strong similarity but weak locality.
- Linux-set, downloaded from the website [19], consists of 900 versions from version 1.1.13 to 2.6.33, and represents the characteristics of small files.
- Full-set1 consists of 380 full backups of 19 researchers'PCs over 20 days, which is reported by Xing et al. [20]. Full-set1 represents the data sets with strong locality and strong similarity.
- Full-set2 was collected from an engineering group consisting of 15 graduate students and was used in [6]. The students in this group ran full or incremental backups independently in a span of 31 days. Full-set2 also represents data sets with strong locality and strong similarity.

Both Linux-set, Full-set1, and Full-set2 are used in previous studies [10], [20], and [6] respectively to evaluate the performance of Extreme Binning, and our use of these data sets resulted in similar and consistent evaluation results with the published studies.

4.2 A Sensitivity Study of SiLo

SiLo's performance is likely influenced by several important factors, including the segment size, the block size, and the cache size (measured by the number of blocks in cache). The mutually interactive nature of similarity and locality in SiLo dictates a good understanding of the relationship between locality and similarity before a thorough performance evaluation is carried out. Thus, we first examine the impact of the SiLo design parameters of block size and segment size on duplicate elimination and time overhead, which is critical for the SiLo locality and similarity algorithms.

Segment size and block size. Fig. 10 shows the percentage of duplicate data eliminated by SiLo approach with different block size and segment size. We observe that the duplicate elimination performance, defined as the percentage of duplicate data eliminated, increases with the block size but decreases with the segment size. This is because the smaller the segment is (e.g., segment size of 512 KB), the more similarity can be exposed and detected, enabling more duplicate data to be removed. On the other hand, the larger the block is (e.g., block size of 512 MB), the more locality of the backup stream will be retained and captured, allowing SiLo to eliminate more than 97 percent of redundant data regardless of the segment size. This is consistent with our motivation and mathematic analysis that combined exploitation of similarity and locality could achieve near-exact duplicate elimination as is shown in Fig. 3 of Section 2.

Although more redundant data can be eliminated by reducing the segment size or filling a block with more segments, as indicated by the results shown in Fig. 10, it results in more accesses to on-disks index and higher RAM usage due to the increased index entries in the SHTable (see Fig. 5). As the time-overhead of deduplication indexing, shown in Fig. 11, clearly suggests, continuously decreasing the segment size or increasing the block size can become counterproductive after a certain point. From Fig. 11, we further find that, for a fixed block size, the time overhead is inversely proportional to the segment size. This is consistent with our intuition that smaller segment size results in more frequent similarity detections for the input segments, which in turn can cause more accesses to on-disk index.

Fig. 11 also shows that there is a knee point for each curve, meaning that for a given segment size and workload the time overhead decreases first and then increases (except Fig. 11c). This can be explained by the fact that, with a very small block size (e.g., 8 MB), there is little locality to be mined, resulting in frequent accesses to on-disk index. With a very large block size (e.g., 512 MB),

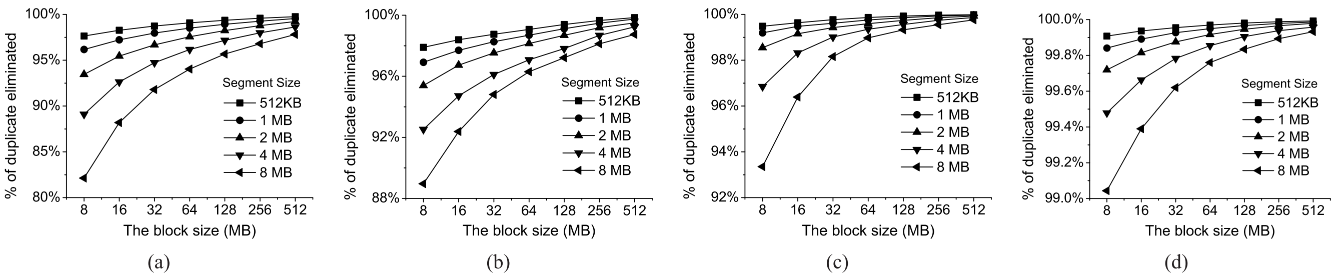


Fig. 10. Percentage of duplicate eliminated as a function of block and segment size. (a) One-set. (b) Inc-set. (c) Linux-set. (d) Full-set1.

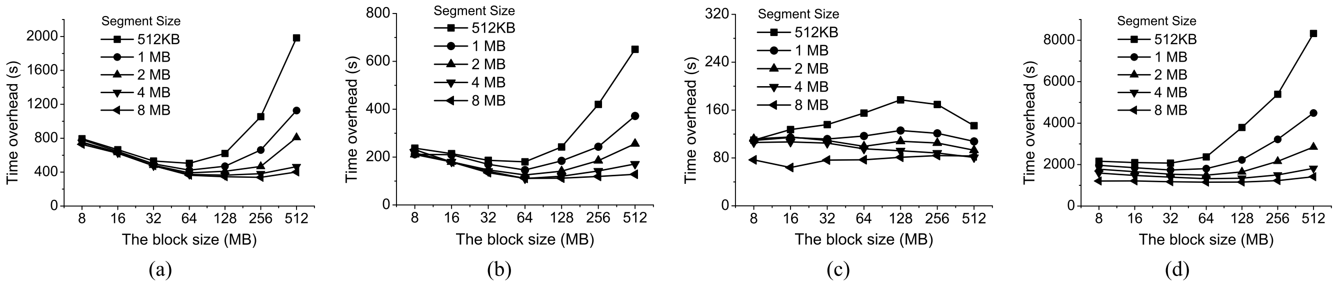


Fig. 11. Time overhead of dedupe indexing as a function of block and segment size. (a) One-set. (b) Inc-set. (c) Linux-set. (d) Full-set1.

SiLo also runs slower because the increased disk accesses for locality exploitation may result in more unrelated segments being read in. The Linux-set is different from other data sets in Fig. 11, because the average size of a Linux version is 110 MB, which enables more related locality to be exploited at the block size of 256 MB.

Size of read cache. Fig. 12 shows the deduplication throughput performance of SiLo with different number of blocks in the read cache under five data sets. Although the deduplication throughput will increase with the number of blocks in the read cache, it results in more RAM overhead to store the cached blocks. With a large number of blocks (e.g., $\text{number} = 16$), the deduplication throughput will increasing slowly and even decrease a little because it may results in more invalid indexing operations in the read cache by SiLo's locality algorithm. Thus the read cache with eight or 16 blocks can be a proper choice for SiLo.

As analyzed from Figs. 10, 11, and 12, there evidently exists an optimum segment size (e.g., 2 MB), an optimum block size (e.g., 256 MB), and an optimum size of the read cache (e.g., 16 blocks), subject to a given workload and deduplication requirement (e.g., duplicate elimination or deduplication throughput). The choices of segment size, block size, and number of blocks can be adjusted by the user's specific requirements, such as larger segment size for higher throughput and lower RAM overhead but less duplicate elimination, and larger block size for more duplicate elimination but lower throughput.

Locality enhanced similarity detection. Fig. 13 shows the duplicate data eliminated respectively by SiLo's similarity approach and SiLo's locality approach under different similarity degree on the Linux data set. The missed portion of duplicate elimination is defined as the difference between the measure achieved by the exact deduplication and that by the SiLo approach. Therefore, the similarity-based deduplication efficiency shown in Fig. 13 heavily depends on the similarity degree of the backup stream which is well consistent with Broder's Theorem (see Section 2.2). The similarity approach often fails to remove large amounts of duplicate data, especially when the backup stream has a low similarity degree. But full exploitation of locality jointly with that of similarity can remove almost all redundant data missed by the similarity detection regardless of similarity degree. Fig. 14 further demonstrates that the combined exploitation of locality and similarity can remove almost all redundant data under all workloads. In fact, only an extremely small amount of duplicate data is missed by SiLo even on the data sets with weak locality and similarity. The results of Figs. 13 and 14 can be compared with Figs. 2a and 2b, then well verify our motivation of similarity and locality in Section 2.

4.3 Comparison Evaluations of SiLo

This section presents evaluation results comparing SiLo with two other state-of-the-art deduplication systems, the similarity-based Extreme Binning system and the locality-based ChunkStash system, by executing the five real-world traces described in Section 4.1 on these three systems. Note that in this evaluation SiLo assumes a block size of 256 MB and a read-cache size of 10 blocks, while SiLo-2 MB and SiLo-4MB represent SiLo with a segment size of 2 and 4 MB respectively. Thus the SiLo-2 MB approach exploits more similarity, as discussed in Section 4.2, while SiLo-4 MB

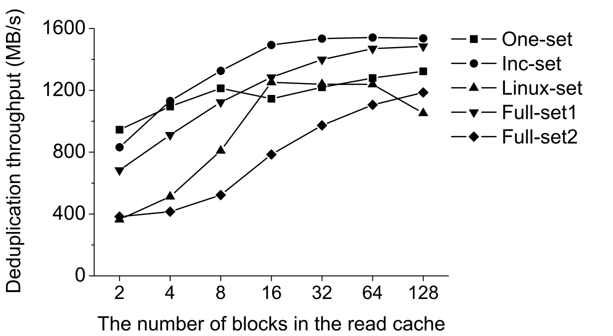


Fig. 12. Throughput of deduplication indexing on five data sets as a function of the read cache size (i.e., the number of blocks).

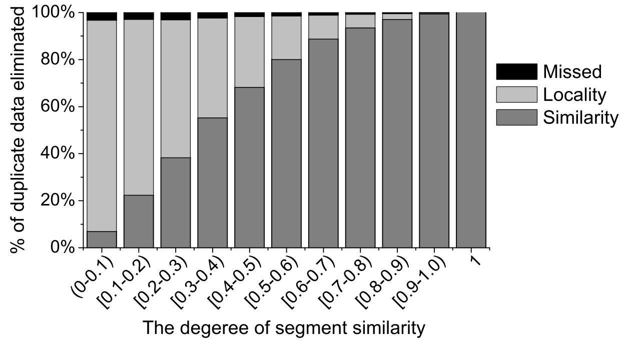


Fig. 13. Percentage of duplicate data eliminated as a function of different similarity degrees on the Linux-set by the similarity-only approach and locality-only approach respectively.

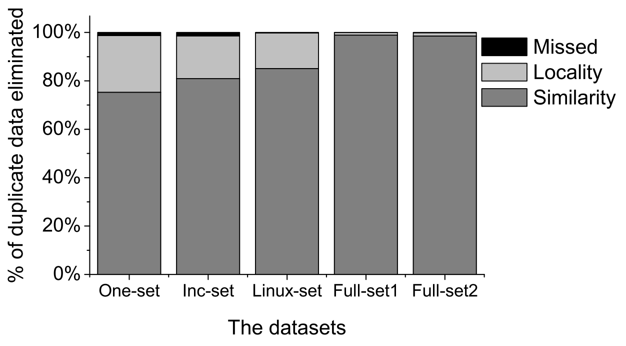


Fig. 14. Percentage of duplicate data eliminated on four data sets by the similarity-only approach and locality-only approach respectively.

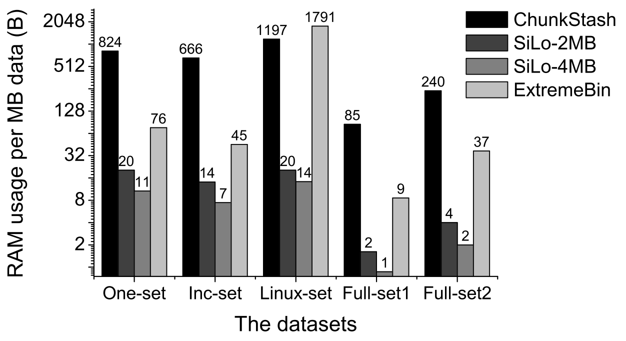


Fig. 16. Comparisons among ChunkStash, SiLo, and Extreme Binning in terms of RAM usage (B: RAM required per MB backup data).

consumes less RAM space for similarity indexing but at the cost of duplicate elimination.

4.3.1 Duplicate Elimination

Fig. 15 shows the duplicate elimination performance of the three systems under the five workloads. Since ChunkStash does the exact deduplication, it eliminates 100 percent of duplicate data. Compared with Extreme Binning that eliminates 71∼99 percent of duplicate data in the five data sets, SiLo removes about 98.5∼99.9 percent of duplicate data. Note that, while Extreme Binning eliminates about 99 percent of duplicate data as expected in Linux-set, Full-set1 and Full-set2 that has strong similarity and locality, it fails to detect almost 30 percent of duplicate data in One-set that has weak locality and similarity, and about 25 percent of duplicate data in Inc-set with weak locality but strong similarity. Although there is strong similarity in Inc-set, Extreme Binning still fails to eliminate a significant amount of duplicate data primarily due to its probabilistic similarity detection that simply chooses one representative fingerprint for each file regardless of the file size.

On the contrary, SiLo-2 MB eliminates 99 percent of duplicate data even in One-set with both weak similarity and locality, and also removes almost 99.9 percent of duplicate data in Linux-set, Full-set1 and Full-set2 with both strong similarity and locality. These results show that SiLo's joint and complementary exploitation of similarity and locality is very effective in detecting and eliminating duplicate data under all workloads, achieving near-complete duplicate elimination.

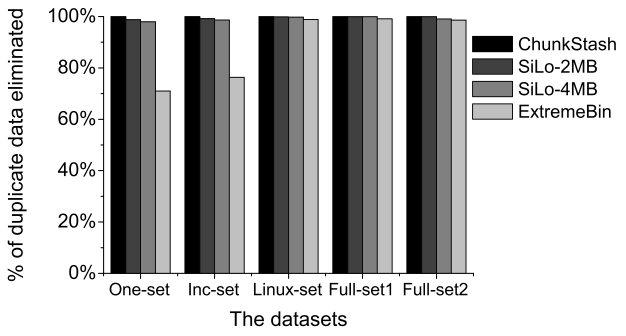


Fig. 15. Comparison among ChunkStash, SiLo, and Extreme Binning in terms of percentage of duplicate data eliminated on the five data sets.

4.3.2 RAM Usage for Deduplication Indexing

Fig. 16 shows the RAM usage for deduplication among these three systems under the five workloads. For Linux-set that has a very large number of small files and small chunks, the highest RAM usage is incurred for both Chunkstash and Extreme Binning. There is also a clear negative correlation between the deduplication factor and the RAM usage for the approximate deduplication systems of SiLo and Extreme Binning on the other four workloads. That is, for One-set that has the lowest deduplication factor, the highest RAM usage is incurred, while for Full-set1 that has the highest deduplication factor, the smallest RAM space is required.

The average RAM usage for ChunkStash is the highest among the three approaches, except for the Linux-set trace, as it does the exact deduplication that needs a large hash table in the memory to put all the indices of chunk fingerprints. Although ChunkStash uses the Cuckoo hash to store the compact key signatures instead of full chunk-fingerprints, it still requires at least 6 bytes for each new chunk. In addition, according to the open-source code of Cuckoo Hash which is used in this paper for ChunkStash evaluation [22], it needs to allocate about two million slots in advance to support one million index entries. Note that using the variant of Cuckoo hash may incur a high load factor but still store at least 6 bytes for each new chunk in the ChunkStash system.

Since only the file similarity index needs to be stored in the RAM, Extreme Binning only consumes about 1/9∼1/15 of the RAM space required of ChunkStash except on the Linux-set where it consumes more RAM usage than ChunkStash due to the extremely large number of small files. However, SiLo-2MB's RAM efficiency allows it to reduce the RAM consumption of Extreme Binning by a factor of 3∼900. The extremely low RAM overhead of SiLo stems from the interplay between its similarity and locality algorithm. On the other hand, the RAM usage for Extreme Binning depends on the average file size of data sets, in addition to the deduplication factor. The smaller the average file size is, the more RAM space Extreme Binning will consume, which is demonstrated in the Linux-set. The RAM usage of SiLo remains relatively stable with the change in average file size in the five traces and is inversely proportional to the deduplication factor of the traces as shown in Fig. 16.

TABLE 2
Comparisons of Several State-of-the-Art Deduplication Approaches in Terms of RAM Usage per PB Unique Data

State-of-the-art Dedupe Approaches	Exact or Approximate	Average Chunk Size	RAM Usage per PB Data
DDFS	Exact	8KB	125GB
Sparse Indexing	Approximate	4KB	85GB
Extreme Binning	Approximate	N/A	300GB
ChunkStash	Exact	8KB	0.75TB
MAD2	Exact	4KB	1TB
HPDS[21]	Approximate	4KB	50GB
SiLo-2MB	Approximate	N/A	30GB
SiLo-4MB	Approximate	N/A	15GB

We assume that average file size is 200 KB, secure fingerprint is SHA-1, and chunks are not compressed.

Table 2 shows the RAM usage in a PB-scale deduplication system for several state-of-the-art approaches. As a representative fingerprint requires about 60 bytes as key index in the memory regardless of the average chunk size, Extreme Binning [10] demands almost 300 GB of RAM space with a mean file size of about 200 KB [14] in a PB-scale deduplication system while SiLo-2 MB and SiLo-4 MB consume about 30 and 15 GB memory by a similarity and locality based deduplication index design. DDFS [4] consumes about 125 GB RAM space for deduplicating 1 PB unique data (one byte per chunk by the Bloom filter) while Sparse Indexing [9] reduces the RAM usage to about 85 GB by a sparse index design with 1/64 sampling. HPDS [21] further reduces the RAM usage to about 50 GB by a progressive sampled index with a sampling rate of 1/101. Both ChunkStash [11] and MAD2 [6] consume almost 1 TB of RAM space to maintain a global index in a PB-scale deduplication system by their cuckoo-hash based and bloom-filter-array based indexing schemes respectively. Thus SiLo uses significantly less RAM space than the above state-of-the-art approaches. As the RAM space is still limited in computer systems, the lower usage of memory means a higher scalability of the SiLo deduplication system.

4.3.3 Deduplication Throughput

Fig. 17 shows a comparison among the three approaches in terms of deduplication throughput, where the throughput is observed to more than double as the average chunk size changes from 5 (e.g., Full-set2) to 10 KB (e.g., One-set). ChunkStash achieves an average throughput of about

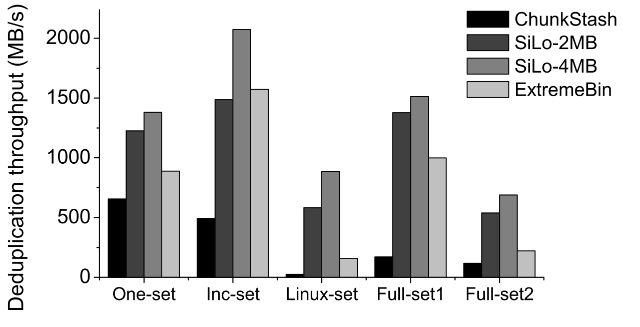


Fig. 17. Comparison among ChunkStash, SiLo, and Extreme Binning in terms of deduplication throughput (MB/sec).

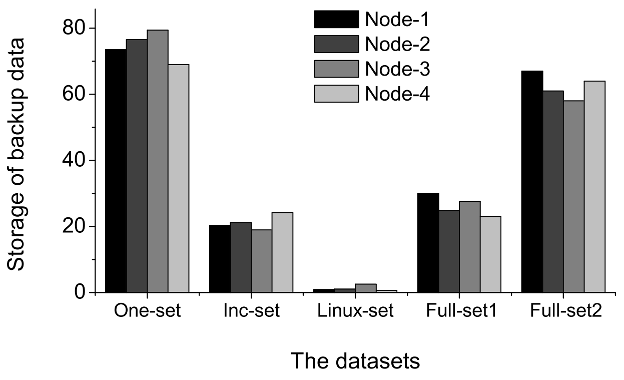


Fig. 18. Size of deduplicated data on each backup node under five real-world workloads when using 4 backup nodes.

292 MB/sec with a range of 24-654 MB/sec on the five data sets. The frequency of accesses to on-disk index by ChunkStash's compact key signatures algorithm on the Cuckoo hash lookup tends to increase with the size of the data set, thus adversely affecting the throughput. Extreme Binning achieves an average throughput of 768 MB/sec with a range of 158-1571 MB/sec on the five data sets, since it only needs to access the disk once per similar-file and eliminates the duplicate files in the memory. As SiLo-2 MB makes at most one disk access per segment, it deduplicates data at an average throughput of 1,042 MB/sec with a range of 538-1486 MB/sec on the five data sets.

Although Extreme Binning runs faster than SiLo-2 MB under Inc-set where many duplicate files exist, it runs much slower in other data sets. Since each bin stores all similar files and it tends to grow in size with the size of data sets. As a result, Extreme Binning will slow down as the size of each bin increases since each similar file must read its corresponding bin in its entirety. In addition, the design of bin fails to exploit the backup-stream locality that helps reduce disk accesses and increase the RAM utilization by preserving the locality layout in RAM.

Therefore, compared with Extreme Binning and ChunkStash, SiLo is shown to provide robust and consistently good deduplication performance, achieving higher throughput, and near-exact duplicate elimination at a much lower RAM overhead.

4.3.4 Load Distribution of SiLo

This section presents evaluation results of SiLo's locality based routing algorithm. Fig. 18 shows how much deduplicated data is managed by each backup node on five data sets. Since only unique chunk contents (non-duplicate) are stored on disk ultimately, the deduplicated data sizes in fact demonstrate the storage loads of real world applications. It is clear that no single node gets overloaded and the deduplicated data of each data set is distributed evenly. The One-set is deduplicated to 530 GB and then stored to four distributed backup nodes with size of 73, 76, 79, and 69 GB respectively. Other data sets show similar trends as the One-set.

The results in Fig. 18 show that the distribution of datablock to backup nodes is not uneven and achieves the performance of well balanced load. This property of SiLo is important to guarantee smooth scale out, improve data reliability and prevent any node from becoming a bottleneck of the overall system performance. Due to the randomness property of hash digest algorithms (e.g., SHA-1), a balanced load can also be achieved by SiLo's routing algorithm when more nodes are added.

5 Related Work

Data deduplication is an essential and critical component of backup/archiving storage systems. We briefly review the work that is most relevant to SiLo to put it in the appropriate perspective. LBFS [8] first proposes the content-defined Chunking algorithm adopting the Rabin fingerprints [23], and applies it to the network file system to reduce transmission of redundant data. Venti [1] employs deduplication in an archival storage system and significantly reduces the storage space requirements. Policroniades and Pratt [3] and Kulkarni et al. [2] compare the performance of several deduplication approaches, such as file-level, fixed-size chunking, and content-based chunking. Besides the above chunking approaches, a lot of efforts have been put into optimizing chunking algorithms, such as Bimodal chunking [13], two threshold two denominators (TTTD) [24], etc.

Recently, increasing attention has been paid to avoiding the fingerprint-lookup disk bottleneck and enabling more efficient, reliable, and scalable deduplication in mass storage systems [6], [7], [10], [15], [25]. DDFS [4] and Sparse Indexing [9] have been elaborated in Section 2.1 as two well-known locality based deduplication approaches. Guo and Efstathopoulos [21] also exploits backup-stream locality with a progressive sampled indexing approach to achieve about 97 percent of deduplication efficiency. ChunkStash [11] stores the chunk fingerprints on an SSD instead of an HDD to accelerate the index-lookup. It preserves the backup-stream locality in the memory to increase the RAM utilization and reduce accesses to on-disk/ssd index. Cuckoo hash [22] is used by ChunkStash to organize the fingerprint index in RAM, which is shown to be more efficient than Bloom filters in DDFS.

The aforementioned locality-based approaches would produce poor performance of deduplication in the case of the data streams with little or no locality [10]. Several earlier studies [17], [26] propose to exploit similarity characteristics for small-scale deduplication of documents in the field of knowledge discovery and database. Kulkarni et al. [2] propose a data reduction approach that breaks files into chunks, deduplicates identical chunks, identifies similar chunks among the remaining ones by a super-fingerprint approach, and delta-encodes the similar chunks to further save space. Aronovich et al. [27] exploit the similarity of backup streams in mass deduplication systems. They divide a data stream into large 16 MB blocks, construct signatures to identify possibly similar blocks and then conduct the byte-by-byte comparison to eliminate duplicate data. Extreme Binning [10] exploits the file similarity for deduplication to apply to non-traditional backup workloads with low-locality.

Compared with the existing work [4], [9], [10], [11], SiLo complementarily makes full use of both similarity and locality in backup streams to further boost data deduplication throughput and minimize the RAM overhead. SiLo is in part inspired by the Cumulus system [28] and Bimodal Chunking algorithm [13]. Cumulus is designed for filesystem backup over the Internet under the assumption of a thin cloud [28]. It proposes the aggregation of many small files to a segment to avoid frequent network transfers of small files in the backup system, and implements a general user-level deduplication. Bimodal Chunking [13] aims to reduce the size of index by exploiting data-stream locality. It merges some contiguous and duplicate chunks, produces a chunk size that is about 2-4 times larger than that of general algorithms, and finds more potential duplicate data among the boundaries of duplicate chunks.

More recently, there have been increasing attentions being paid to the load distribution of deduplication based storage systems. HYDRAstor [25] performs the deduplication at a large-chunk (64 KB) granularity and distributes data at the chunk level using distributed hash tables. Super-chunk based data routing [12] exploits data similarity to direct data routing at the super-chunk level while Extreme Binning [10] mines file similarity for distributed storage. Compared with these recent approaches, SiLo exploits both similarity and locality to obtain a well-balanced load while achieving near-exact deduplication at an extremely low memory overhead.

As deduplication techniques are compute-intensive due to their tasks of Rabin-based chunking and secure-signature based fingerprinting, hash calculations are increasingly recognized as a potential bottleneck in high performance deduplication systems. Guo and Efstathopoulos. [21] propose an event-driven, multi-threaded client-server interaction model to improve deduplication throughput. Shredder [29] makes full use of GPU's computational power to accelerate the compute-intensive primitives of chunking and fingerprinting in deduplication based storage systems while P-Dedupe [30] proposes an approach of composing pipelined and parallel computations of data deduplication to run in multicore processors.

6 Conclusion and Future Work

To address the scalability of data deduplication and meet increasing size of data storage scale in mass storage system, we present SiLo, a similarity-locality based deduplication system that exploits both similarity and locality in backup streams to achieve higher deduplication throughput, well balanced load, and near-complete duplicate elimination at an extremely lower RAM overhead than existing state-of-the-art approaches. Results from our prototype evaluation driven by real-world data sets show that the SiLo similarity algorithm significantly reduces the RAM usage, its locality algorithm helps eliminate most of the duplicate data that is missed by the probabilistic similarity detection, and its load distribution algorithm obtains a well balanced load.

As our future work, we plan to study SiLo' indexing scheme in other deduplication applications such as cloud storage and primary storage that is very different from backup/archiving storage because of the relatively weak locality and similarity in workloads and high sensitivity to latency when implementing inline deduplication. Moreover, since the read/deletion operations are intuitively more frequent and important in primary storage deduplication [31], the research on read performance and garbage collection on the deduplicated data will be an important and promising research topic in deduplication based primary storage systems.

ACKNOWLEDGMENTS

This work was supported in part by National Basic Research 973 Program of China under Grant 2011CB302301; National Science Fund for Distinguished Young Scholars of China under Grant 61025008; National Natural Science Foundation of China (NSFC) under Grant 61173043 and 61232004; The National High Technology Research and Development Program ("863" Program) of China under Grant No. 2013AA013203; US National Science Fondation (NSF) under Grants NSF-IIS-0916859, NSF-CCF-0937993, NSF CNS-1116606, and NSF-CNS-1016609. Dan Feng is the corresponding author. The preliminary manuscript appeared in the proceedings of USENIX Annual Technical Conference (ATC), 2011.

REFERENCES

[1] S. Quinlan and S. Dorward, "Venti: A New Approach to Archival Storage," *Proc. First USENIX Conf. File and Storage Technologies*, 2002.

[2] P. Kulkarni, F. Douglis, J. LaVoie, and J. Tracey, "Redundancy Elimination within Large Collections of Files," *Proc. USENIX Ann. Technical Conf.*, pp. 59-72, 2004.

[3] C. Policroniades and I. Pratt, "Alternatives for Detecting Redundancy in Storage Systems Data," *Proc. Ann. Conf. USENIX Ann. Technical Conf.*, 2004.

[4] B. Zhu, K. Li, and H. Patterson, "Avoiding the Disk Bottleneck in the Data Domain Deduplication File System," *Proc. Sixth USENIX Conf. File and Storage Technologies*, 2008.

[5] K. Jin and E. Miller, "The Effectiveness of Deduplication on Virtual Machine Disk Images," *Proc. SYSTOR 2009: The Israeli Experimental Systems Conf.*, pp. 1-12, 2009.

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

[7] D. Meyer and W. Bolosky, "A Study of Practical Deduplication," *Proc. Ninth USENIX Conf. File and Storage Technologies*, 2011.

[8] A. Muthitacharoen, B. Chen, and D. Mazieres, "A Low-Bandwidth Network File System," *Proc. 18th ACM Symp. Operating Systems Principles*, pp. 174-187, 2001.

[9] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, "Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality," *Proc. Seventh Conf. File and Storage Technologies*, pp. 111-123, 2009.

[10] D. Bhagwat, K. Eshghi, D. Long, and M. Lillibridge, "Extreme Binning: Scalable, Parallel Deduplication for Chunk-Based File Backup," *Proc. IEEE Int'l Symp. Modeling, Analysis & Simulation of Computer and Telecomm. Systems*, pp. 1-9, 2009.

[11] B. Debnath, S. Sengupta, and J. Li, "Chunkstash: Speeding Up Inline Storage Deduplication Using Flash Memory," *Proc. 2010 USENIX Conf. USENIX Ann. Technical Conf.*, 2010.

[12] W. Dong, F. Douglis, K. Li, H. Patterson, S. Reddy, and P. Shilane, "Tradeoffs in Scalable Data Routing for Deduplication Clusters," *Proc. Ninth USENIX Conf. File and Storage Technologies*, 2011.

[13] E. Kruus, C. Ungureanu, and C. Dubnicki, "Bimodal Content Defined Chunking for Backup Streams," *Proc. Eighth USENIX Conf. File and Storage Technologies*, 2010.

[14] N. Agrawal, W. Bolosky, J. Douceur, and J. Lorch, "A Five-Year Study of File-System Metadata," *ACM Trans. Storage*, vol. 3, no. 3, article 9, 2007.

[15] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu, "Characteristics of Backup Workloads in Production Systems," *Proc. 10th USENIX Conf. File and Storage Technologies*, 2012.

[16] A. Broder, "On the Resemblance and Containment of Documents," *Proc. Compression and Complexity of Sequences*, 1997.

[17] D. Bhagwat, K. Eshghi, and P. Mehra, "Content-Based Document Routing and Index Partitioning for Scalable Similarity-Based Searches in a Large Corpus," *Proc. 13th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, pp. 105-112, 2007.

[18] Y. Tan, H. Jiang, D. Feng, L. Tian, Z. Yan, and G. Zhou, "SAM: A Semantic-Aware Multi-Tiered Source De-Duplication Framework for Cloud Backup," *Proc. IEEE 39th Int'l Conf. Parallel Processing*, pp. 614-623, 2010.

[19] "Linux download, ftp://ftp.kernel.org/," 2014.

[20] Y. Xing, Z. Li, and Y. Dai, "PeerDedupe: Insights into the Peer-Assisted Sampling Deduplication," *Proc. IEEE 10th Int'l Conf. Peer-to-Peer Computing (P2P)*, 2010.

[21] F. Guo and P. Efstathopoulos, "Building a High-Performance Deduplication System," *Proc. 2011 Conf. USENIX Ann. Technical Conf.*, 2011.

[22] R. Pagh and F.F. Rodler, "Cuckoo Hashing," *J. Algorithms*, vol. 51, no. 2, pp. 122-144, 2004.

[23] A. Broder, "Some Applications of Rabin*a*s Fingerprinting Method," *Sequences II: Methods in Comm., Security, and Computer Science*. Springer, 1993.

[24] K. Eshghi and H.K. Tang, "A Framework for Analyzing and Improving Content-Based Chunking Algorithms," Technical Report HPL-2005-30R1 HP Labs, 2005.

[25] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki, "HYDRAstor: A Scalable Secondary Storage," *Proc. Seventh Conf. File and Storage Technologies*, pp. 197-210, 2009.

[26] U. Manber et al., "Finding Similar Files in a Large File System," *Proc. USENIX Winter Technical Conf.*, pp. 1-10, 1994.

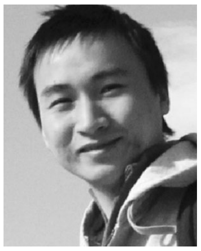
[27] L. Aronovich, R. Asher, E. Bachmat, H. Bitner, M. Hirsch, and S. Klein, "The Design of a Similarity Based Deduplication System," *Proc. SYSTOR 2009: The Israeli Experimental Systems Conf.*, 2009.

[28] V. Michael, S. Stefan, and M. Geoffrey, "Cumulus: Filesystem Backup to the Cloud," *Proc. Seventh USENIX Conf. File and Storage Technologies*, 2009.

[29] P. Bhatotia, R. Rodrigues, and A. Verma, "Shredder: GPU-Accelerated Incremental Storage and Computation," *Proc. 10th USENIX Conf. File and Storage Technologies*, 2012.

[30] W. Xia, H. Jiang, D. Feng, L. Tian, M. Fu, and Z. Wang, "P-Dedupe: Exploiting Parallelism in Data Deduplication System," *Proc. IEEE Seventh Int'l Conf. Networking, Architecture and Storage (NAS)*, pp. 338-347, 2012.

[31] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti, "iDedup: Latency-Aware, Inline Data Deduplication for Primary Storage," *Proc. 10th USENIX Conf. File and Storage Technologies*, 2012.



Wen Xia is currently working toward the PhD degree majoring in computer architecture from the Huazhong University of Science and Technology, Wuhan, China. His current research interests include data deduplication, backup storage system, delta compression, and data similarity detection. He has more than five publications in journals and international conferences including USENIX ATC, DCC, NAS.

Hong Jiang received the BSc degree in computer engineering in 1982 from the Huazhong University of Science and Technology, Wuhan, China, the MASc degree in computer engineering in 1987 from the University of Toronto, Canada, and the PhD degree in computer science in 1991 from Texas A&M University, College Station, Texas. Since August 1991, he has been at the University of Nebraska-Lincoln, Lincoln, Nebraska, where he is Willa Cather professor of Computer Science and Engineering. His current research interests include computer architecture, computer storage systems and parallel I/O, high-performance computing, big data computing, cloud computing, performance evaluation. He is an associate editor of the *IEEE Transactions on Parallel and Distributed Systems*. He has more than 200 publications in major journals and international conferences in these areas, including IEEE-TC, IEEE-TPDS, ACM-TACO, JPDC, ISCA, MICRO, USENIX ATC, FAST, LISA, ICDCS, IPDPS, MIDDLEWARE, OOPLAS, ECOOP, SC, ICS, HPDC, ICPP. He is a senior member of the IEEE and a member of the ACM.

Dan Feng received the BE, ME, and the PhD degrees in computer science and technology in 1991, 1994, and 1997, respectively, from the Huazhong University of Science and Technology (HUST), China. She is a professor and vice dean of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, massive storage systems, and parallel file systems. She has more than 100 publications in major journals and international conferences, including *IEEE Transactions on Computers*, *IEEE Transactions on Parallel and Distributed Systems*, *ACM Transactions on Storage*, *Journal of Computer Science and Technology*, FAST, USENIX ATC, ICDCS, HPDC, SC, ICS, IPDPS, and ICPP. She has served on the program committees of multiple international conferences, including SC 2011, 2013 and MSST 2012. She is a member of the IEEE and the ACM.

Yu Hua received the BE and PhD degrees in computer science from Wuhan University, China, in 2001 and 2005, respectively. He is an associate professor at the Huazhong University of Science and Technology, China. His research interests include computer architecture, cloud computing, and network storage. He has more than 50 papers to his credit in major journals and international conferences including *IEEE Transactions on Computers*, *IEEE Transactions on Parallel and Distributed Systems*, USENIX ATC, INFOCOM, SC, ICDCS, ICPP, and MASCOTS. He has been on the program committees of multiple international conferences, including INFOCOM and ICPP. He is a senior member of the IEEE, and a member of the ACM and USENIX.