SiLo: A Similarity-Locality based Near-Exact Deduplication Scheme with Low RAM Overhead and High Throughput

Wen Xia [†]
wx.hust@gmail.com

Hong Jiang [‡]
jiang@cse.unl.edu

Dan Feng [†] ✉
dfeng@hust.edu.cn

Yu Hua [†,‡]
csyhua@hust.edu.cn

[†] School of Computer, Huazhong University of Science and Technology, Wuhan, China
Wuhan National Lab for Optoelectronics, Wuhan, China
[‡]Dept. of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE, USA

Abstract

Data Deduplication is becoming increasingly popular in storage systems as a space-efficient approach to data backup and archiving. Most existing state-of-the-art deduplication methods are either locality based or similarity based, which, according to our analysis, do not work adequately in many situations. While the former produces poor deduplication throughput when there is little or no locality in datasets, the latter can fail to identify and thus remove significant amounts of redundant data when there is a lack of similarity among files. In this paper, we present SiLo, a near-exact deduplication system that effectively and complementarily exploits similarity and locality to achieve high duplicate elimination and throughput at extremely low RAM overheads. 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 locality in the backup stream by grouping contiguous segments into blocks to capture similar and duplicate data missed by the probabilistic similarity detection. By judiciously enhancing similarity through the exploitation of locality and vice versa, the SiLo approach is able to significantly reduce RAM usage for index-lookup and maintain a very high deduplication throughput. Our experimental evaluation of SiLo based on real-world datasets shows that the SiLo system consistently and significantly outperforms two existing state-of-the-art system, one based on similarity and the other based on locality, under various workload conditions.

1 Introduction

As the amount of the important data that needs to be digitally stored grows explosively to a worldwide storage crisis, data deduplication, a space-efficient method, has gained increasing attention and popularity in data storage. It splits files into multiple chunks that are each uniquely identified by a 20-byte SHA-1 hash signature, also called a fingerprint [21]. It removes duplicate chunks by checking their fingerprints, which avoids a byte-by-byte comparison. Data deduplication not only reduces the storage space overheads, but also minimizes the network transmission of redundant data in the network storage system [19].

One of the main challenges for centralized backup services based on deduplication is the scalability of fingerprint-index search. For example, to backup a dataset of 800TB and assuming an average chunk size of 8KB, at least 2TB of fingerprints have to be generated, which will be too large to be stored in the memory. Since the access to on-disk index is at least 1000 times slower than that to RAM, the frequent accesses to on-disk fingerprints are not acceptable for backup services and have become the main performance bottleneck of such deduplication systems.

Most of the existing solutions aim to make the full use of RAM, 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 [26] and ChunkStash [8]. 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. Exploitation of 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 [3] that exploits 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 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 with little or no locality (e.g., when users only do the incremental backup). On the other hand, the similarity-based Extreme Binning approach is shown to fail to find significant amount of duplicate data in datasets with little or no file similarity (e.g., when the files are edited frequently). Fortunately, our preliminary study indicates that the judicious exploitation of locality can compensate for the lack of similarity in datasets, 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.

To this end, we propose SiLo, a scalable and low-overhead near-exact deduplication system, to overcome the aforementioned shortcomings of existing state-of-the-art schemes. The main idea of SiLo is to consider both similarity and locality in the backup stream simultaneously. Specifically, we expose and exploit more similarity by grouping strongly correlated small files into a segment and segmenting large files, and leverage locality in the backup stream by grouping contiguous segments into blocks to capture similar and duplicate data missed by the probabilistic similarity detection. The main contributions of this paper include:

- SiLo proposes a new similarity algorithm that groups many small strongly-correlated files into a segment or segments a large file to better expose and exploit their similarity characteristics. This grouping of small files results in much smaller similarity index for segments than chunk index, which can easily fit into RAM for a much larger dataset. The segmenting of large files can expose and thus extract more similarity characteristics so as to remove duplicate data with a higher probability.

- SiLo proposes an effective approach to mining the locality characteristics to capture similar and duplicate data missed by the probabilistic similarity detection by grouping multiple contiguous segments into a block, the basic cache and write-buffer unit, while preserving the spatial locality inherent in the backup stream on the disk. By keeping the similarity index and preserving spatial locality of backup streams in RAM (i.e., hash table and locality cache), SiLo is able to remove large amounts of redundant data, dramatically reduce the numbers of accesses to on-disk index, and substantially increase the RAM utilization.

- Our experimental evaluation of SiLo, based on real-world datasets, shows that the SiLo system consistently and significantly outperforms two existing state-of-the-art systems, the similarity-based Extreme Binning system and the locality-based ChunkStash system, under various workload conditions. According to our evaluations on duplicate elimination, SiLo can remove 1%$\sim$28% more redundant data than Extreme Binning and only 0.1%$\sim$1% less than the exact-deduplicating ChunkStash. Our evaluations on deduplication throughput (MB/sec) suggest that SiLo outperforms ChunkStash by a factor of about 3 and Extreme Binning by a factor of about 1.5. On the RAM utilization for the same datasets, SiLo consumes a RAM capacity that is only 1/41$\sim$1/60 and 1/3$\sim$1/90 respectively of that consumed by ChunkStash and Extreme Binning.

The rest of the paper is organized as follow. Section 2 presents background and motivation for this research. Section 3 describes the architecture and the design of the SiLo system. Section 4 presents our experimental evaluation of SiLo and discusses the results, including the comparisons with the state-of-the-art ChunkStash and Extreme Binning systems. Section 5 gives an overview of related work, and Section 6 draws conclusions and outlines future work.

2 Background and Motivation

In this section, we first provide the necessary background for our SiLo research by introducing the existing acceleration approaches for data deduplication, and then motivate our research by analyzing our observations based on extensive experiments on locality- and similarity-based deduplication acceleration approaches under real-word workloads.

2.1 Deduplication Acceleration Approaches

Previous studies have shown that the main challenge facing data deduplication lies in the on-disk index-lookup

bottleneck [26, 15, 8]. As the size of dataset 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 datasets. This can result in frequent disk accesses for fingerprint-index lookups, thus severely limiting the throughput of the deduplication system. 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.

The locality in the former refers to the observation that files, say A and B (thus their data chunks), in a backup stream appear in approximately the same order throughout multiple full backups with a high probability. DDFS [26] makes full use of this locality characteristic by storing the chunks in the order of the backup stream on the disk and preserving the locality in the RAM. It significantly reduces accesses to the on-disk index by increasing the hit ratio in the RAM. 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, but at the cost of significant RAM overhead. Sparse Indexing [15] improves this method by sampling index instead of using Bloom filters. It uses less than half of the RAM capacity of DDFS. As a novel content-defined chunking algorithm, Bimodal [14] suggests that the neighboring data of duplicate chunks should be assumed to be good deduplication candidates due to backup-stream locality, which can be exploited to maximize the chunk size.

Nevertheless, all these approaches still produce unacceptable performance in face of very large datasets with little or no locality. The similarity-based approaches are proposed to exploit the similar characteristics in backup streams to minimize the chunk-lookup index in the memory. For example, Extreme Binning [3] exploits the similarity among files instead of locality, allowing it to make only one disk access for chunk lookup per file. It significantly reduces the RAM usage by storing only the similarity-based index in the memory. But it often fails to find significant amounts of redundant data when similarity among files is either lacking or weak. 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.

2.2 Small Files and Large Files

Our experimental observations, as well as intuition, suggest that the deduplication of small files can be very space and time consuming. A file system typically contains a very large number of small files [1]. Since the small files (e.g., $\leq$64KB) usually only take up a small fraction (e.g., $\leq$20%) of the total space of a file system but account for a large percentage (e.g., $\geq$80%) of the number of files, the chunk-lookup index for small files will be disproportionally large and likely out of memory. Consequently, the inline deduplication [26, 15] of small files will tend to be very slow and inefficient because of the more frequent accesses to the on-disk index for chunk lookup and the higher network-protocol costs between the client and the server.

This problem of small files can be addressed by grouping many highly correlated small files into a segment. We consider files with the logic sequence within the same parent directory to be highly correlated and thus similar. We exploit the similarity and the locality of a group (i.e., segment) of adjacent small files rather than one individual file or chunk. As a result, at most one access to on-disk index is needed per segment instead of per file or per chunk. The segmenting approach can also minimize the network costs by avoiding the frequent inline interactions per file.

A typical file system also contains many large files (e.g., $\geq$2MB) that only account for a small fraction (e.g., $\leq$20%) of total number of files but occupy a very large percentage (e.g., $\geq$ 80%) of the total space [1]. Obviously, these large files are an important consideration for a deduplication system due to their high space-capacity and bandwidth/time requirements in the backup process. When a large file is being deduplicated inline, the server must often wait for a long time for the chunking and hashing processes, resulting in low efficiency of the deduplication pipeline. In addition, 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, our SiLo approach divides a large file into many small segments to better expose similarity among large files while increasing the efficiency of the deduplication pipeline. More specifically, the probability that file S_1 and file S_2 share the same representative fingerprint is highly dependent on their similarity degree according to Broder's theorem [5]:

Theorem 1: 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) denote the smallest element of the set of integers S. Then:

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

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

$$\Pr[\min(H(S_1) = \min(H(S_2)))] = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} \ll$$
$$\Pr[\min(H(S_{11}) = \min(H(S_{21})))] \cup \cdots \cup \Pr[\min(H(S_{1n}) = \min(H(S_{2n})))]$$
$$= \bigcup_{i=1}^{n} \Pr[\min(H(S_{1i}) = \min(H(S_{2i})))]$$
$$= 1 - \bigcap_{i=1}^{n} \Pr[\min(H(S_{1i}) \neq \min(H(S_{2i})))]$$
$$= 1 - \prod_{i=1}^{n} (1 - \frac{|S_{1i} \cap S_{2i}|}{|S_{1i} \cup S_{2i}|})$$

As files S_1 and S_2 are segmented into $S_{11} \sim S_{1n}$ and $S_{21} \sim S_{2n}$ respectively, the detection of similarity between S_1 and S_2 is determined by the union of the probabilities of detections of similarity 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 or very few of their respective segments are detected as being similar.

2.3 Similarity and Locality

Now we further analyze the relationship between similarity and locality with respect to backup streams. As mentioned earlier, chunk locality can be exploited to store and prefetch groups of contiguous chunks that are likely to be accessed together with a high probability in the backup stream, while files' similarity may be mined so that the similarity characteristics instead of the whole sets of fingerprints of files, are indexed to minimize the index size in the memory. The exploitation of locality maximizes the RAM utilization to improve the throughput but can cause RAM overflows and frequent accesses to on-disk index when datasets lack or are weak in locality.

The similarity-based approaches minimize the RAM usage at the cost of potentially missing large amounts of redundant datawhich is dependent on the similarity degree of the backup stream. We have examined the similarity degree and the duplicate-elimination measure of our similarity-only deduplication approach on four datasets, as shown in Figure 1 and Figure 2. The four datasets represent one-backups, incremental-backups, Linux-versions and full-backups respectively, whose characteristics will be detailed in Section 4.

The similarity degree is computed by our similarity detection on the Linux dataset as: Simi(S_{input}) = Max ($\mid S_{input} \cap S_i \mid$)/$|S_{input}|$,($S_i \in S_{store}$, Simi($S_{input}$) $\in$ [0,1]). Thus, the similarity degree "1" signifies that the input segment is completely duplicate and the similarity degree "0" states that the segment is detected to match no other segments at all by our similarity-only approach.

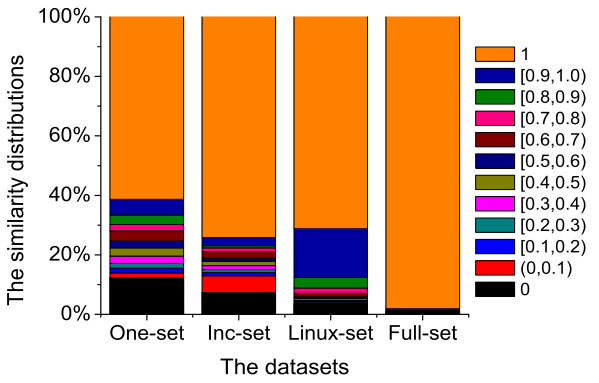


Figure 1: The distribution of the segment similarity on four datasets by our similarity-only approach. It can be used to describe the similarity characteristics of datasets. A large proportion of data with low similarity degrees is observed here.

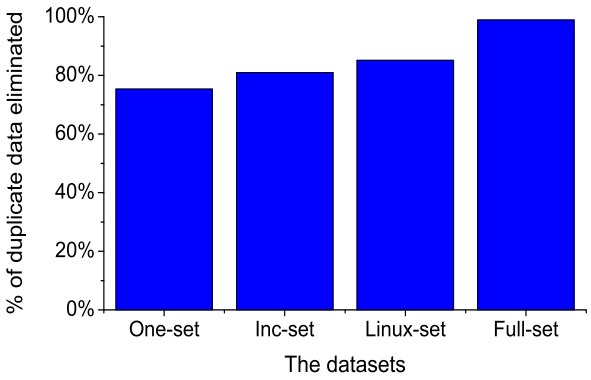


Figure 2: Percentage of duplicate data eliminated by our similarity-only deduplication approach.

Figure 3 further examines the duplicate elimination missed by the similarity approach on the Linux dataset. The missed portion of duplication elimination is defined as the difference between the measure achieved by the exact deduplication and that by the similarity-based deduplication. Therefore, Figure 3 shows that the similarity-based deduplication efficiency is heavily dependent on the similarity degree of the backup stream which is well consistent with Broder's Theorem in (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.

Inspired by Bimodal [14], 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 lo-

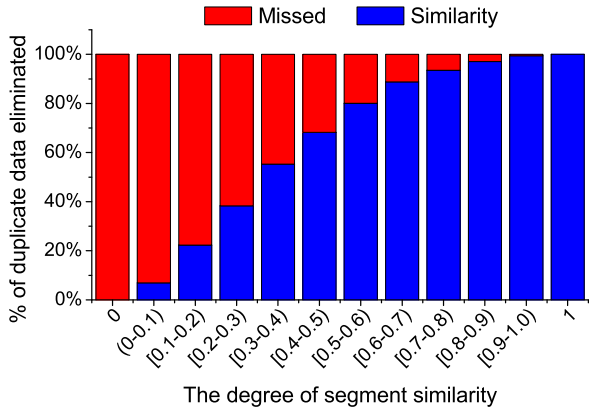


Figure 3: Percentage of duplicate data eliminated as a function of different similarity degree on the Linux dataset by our similarity-only deduplication approach.

cality in conjunction with similarity by grouping multiple contiguous segments in a backup stream into a block. While this exploitation of locality helps find more potential deduplication candidates by detecting similar segments' adjacent segments in a block, it also reduces the accesses to on-disk index to improve the deduplication throughput.

Now we analyze 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:

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

Assume that the value a follows a uniform distribution in the range [0,1] (It may be much more complicated in the real world datasets), the expected value of duplicate elimination can be further calculated under the aforementioned assumption as:

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

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 can remove more than 99% of duplicate data when $N > 99$. Thus the combined exploitation of similarity and locality makes it possible to achieve the near-complete duplicate elimination (recall that exact deduplication achieves complete duplicate elimination) and requires at most one disk access per segment (a group of chunks or small files) rather than one access per chunk (as in locality-based approaches) or per file (as in similarity-based approaches), thus avoiding the disk bottleneck of data deduplication. In addition, the throughput of the deduplication system also tends to be improved by reducing the expensive accesses to on-disk index. As a result, our SiLo approach, through its judicious and joint exploitation of locality and similarity, is able to significantly improve the overall performance of the deduplication system as demonstrated in Section 4.

3 Design and Implementation

SiLo is designed for large-scale and disk-inline backup storage systems. In this section, we will first describe the architecture of SiLo, followed by detailed discussions of its design and implementation issues.

3.1 System Architecture Overview

As depicted in Figure 4, the SiLo architecture consists of four key functional components, namely, File Deamon (FD), Deduplication Server (DS), Storage Server (SS), and Backup Server (BS), which are distributed in the datacenters 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.

- File Deamon is a deamon program providing a functional interface (e.g., backup/restore) in users' computers. It is responsible for gathering backup datasets 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.

- The main function of Deduplication Server is to store and look up all fingerprints of files and chunks.

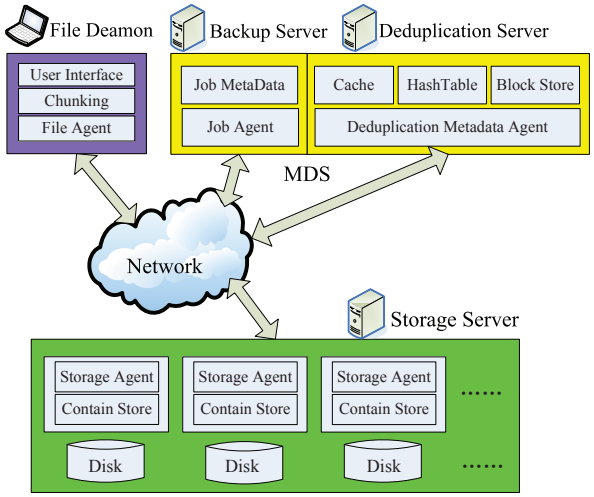


Figure 4: The SiLo system architecture.

- 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 the locality hash table (LHTable), the 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 the backup stream, as shown in Figure 5.

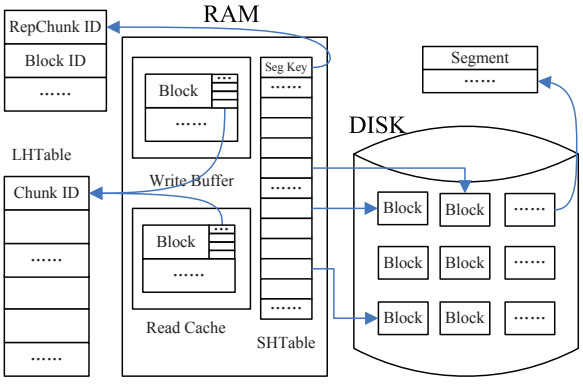


Figure 5: Data structures of Deduplication Server.

The notion of segment is used to exploit the similarity of the 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 index and filter out duplicate chunks. Note that, since this paper mainly aims at improving the performance of accessing on-disk fingerprints in the deduplication system, all write/read operations in this paper are performed in the form of writing/reading chunks' fingerprints rather than the real backup data.

3.2 Similarity Algorithm

As mentioned in Section 2.3, SiLo exploits similarity and locality jointly. 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 Figure 6. Thus, segments are the atomic building units of a block that is in turn the atomic unit of the write buffer and the read cache.

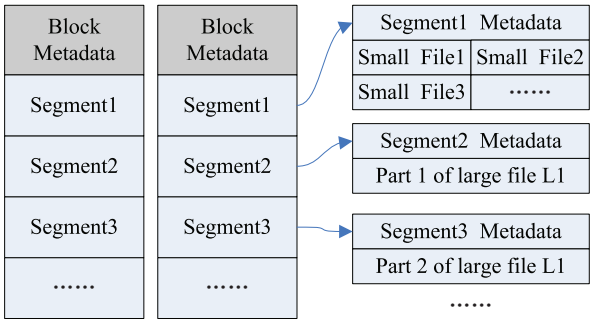


Figure 6: Data structure of the SiLo similarity algorithm.

As a salient feature of SiLo, the SiLo similarity algorithm is implemented in File Deamon, which structures data from backup streams into segments according to the following three principles.

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

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.

SiLo employs the method of representative fingerprinting [3] 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

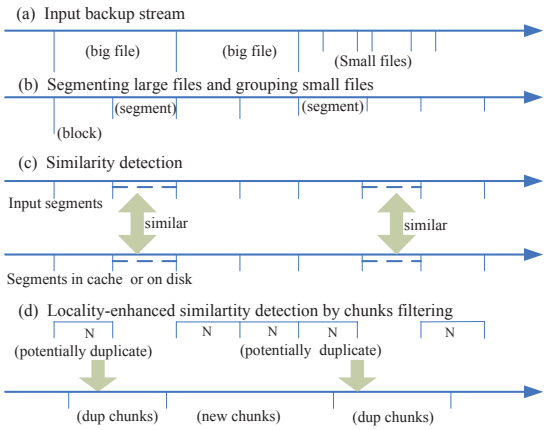


Figure 7: The workflow of the locality algorithm: it helps detect more potentially duplicate chunks that are missed by the similarity algorithm.

files taking up disproportionally large RAM space. For example, assuming an average segment size of 2MB and an average chunk or small file size of 8KB, 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 2MB segment of backup data, which is considered economic, a 1TB backup stream only needs 30MB similarity-index for deduplication that can easily fit in the memory.

3.3 Locality Algorithm

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. Since block is also the minimal write/read unit of the 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 in the backup stream. By exploiting the inherent locality in backup streams, the block-based SiLo locality algorithm is able to eliminate more duplicate data.

Figure 7 shows the workflow of the locality algorithm. According the locality characteristic of backup streams, if input segment S_{1k} in block B_1 is determined to be similar to segment S_{2k} by hitting in the 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.

When SiLo reads the blocks from disk by the similarity detection, it puts the recently accessed block into 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. Since it is at the block level where locality is preserved and exploited, the block size is an important system parameter that affects the system performance such as duplicate elimination and throughput. 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. 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 Figure 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.

3.4 Cache and RAM Considerations

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

Since users of file systems tend to duplicate files or directories under the same directories, a significant amount of duplicate data can be eliminated by detecting the duplication in the 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 can be good deduplication candidates.

The largest portion of RAM in the SiLo system is occupied by the similarity hash table (i.e., SHTable shown in Figure 5). Assuming an average segment size of 2MB and a primary-key size of 60B, the SiLo SHTable requires 300MB for an average backup data of 10 TB. Thus the RAM usage for the cache becomes negligibly small as the data size further increases.

3.5 SiLo Workflow

To put things together and in perspective, Figure 8 shows the main workflow of the SiLo deduplication process. For an incoming backup stream, SiLo goes through the following key steps:

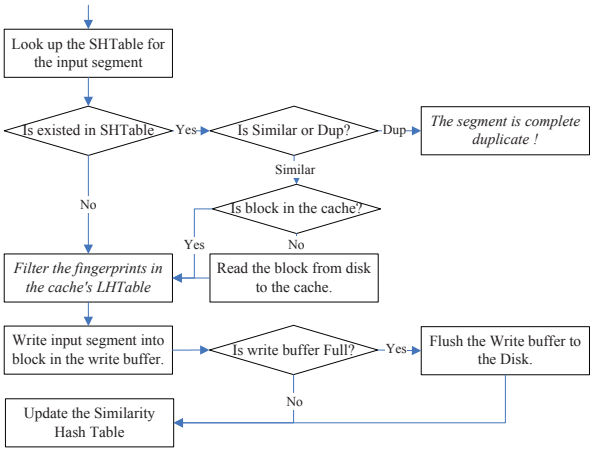


Figure 8: The SiLo deduplication workflow.

1. 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.

2. Each newly generated segment S_{new} is checked against SHTable for similarity detection. If the new segment 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, B_{bk} is read from the disk to the read cache, where a block is replaced in the FIFO order if the cache is full. If S_{new} misses in SHTable, it is then checked against recently accessed blocks in the read cache for potentially similar segment in one of the cached blocks (B_{bk}).

3. The duplicate chunks in S_{new} are eliminated by checking LHTable of B_{bk} in the read cache. Then the chunks in the neighbouring segments of S_{new} in the backup stream are filtered by the locality-enhanced similarity detection (i.e., these chunks are checked against LHTable of B_{bk} for possible duplication).

4. After chunk filtering and constructing a new and non-duplicate block B_{new} from the backup stream, 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.

As demonstrated in Section 4, SiLo is able to minimize both the time and space overheads of indexing fingerprints while maintaining a duplicate elimination performance comparable to exact deduplication methods such as ChunkStash.

4 Evaluation

In order to evaluate SiLo, we have implemented a prototype of SiLo that allows us to examine several important design parameters to provide useful insights. We compare SiLo with the similarity-based and locality-based state-of-the-art approaches Extreme Binning and ChunkStash in the key deduplication metrics of duplicate elimination, RAM usage and throughput. The evaluation is driven by four real-world traces collected from real backup datasets that represent different workload characteristics.

4.1 The Experimental Setup

We use a standard server configuration to evaluate and compare the inline deduplication performances of the SiLo, ChunkStash and Extreme Binning approaches running on a Linux environment. The hardware configuration includes a quad-core CPU running at 2.4GHz, with a 4GB RAM, 2 gigabit network interface cards, and two 500GB 7200rpm 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 [8]. 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 [3]. 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 space overhead of 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

Feature	One-set	Inc-set	Linux	Full-set
Total size	530GB	251GB	101 GB	2.51TB
Total files	3.5M	0.59M	8.8M	11.3M
Total chunks	51.7M	29.4M	16.9M	417.6M
Avg.chunk size	10KB	8KB	5.9KB	6.5KB
Dedupe factor	1.7	2.7	19	25
Locality	weak	weak	strong	strong
Similarity	weak	strong	strong	strong

Table 1: Workload characteristics of the four traces used in the performance evaluation. All use SHA-1 for chunk fingerprints and the content-based chunking algorithm. The deduplication factor is defined as the Totalsize /(Totalsize - Dedupsize) ratio.

backup throughput in that it does not measure the rate at which the backup data is transferred and stored.

Four 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 four traces are collected from real-world datasets of One-backup, Incremental-backup, Linux-version and Full-backup respectively.

The One-set trace was collected from 15 graduate students of our research group. To obtain traces from this backup dataset, we have built a deduplication analysis tool that crawls the backup directory, and generates the sequences of chunk and file hashes for traces. Since we obtain only one full backup for this group, this trace has weak locality and weak similarity. The Inc-set is a subset of the trace reported by Tan et al. [24] 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 251GB data. Therefore, Inc-set represents datasets with strong similarity but weak locality.

Linux-set, downloaded from the website [16], consists of 900 versions from version 1.1.13 to 2.6.33, and represents the characteristics of small files. Full-set consists of 380 full backups of 19 researchers' PCs, which is also reported by Xing et al. [25] and can be downloaded from the website [10]. Full-set represents datasets with strong locality and strong similarity. Both Linux-set and Full-set are used in [25] and [3] to evaluate the performance of Extreme Binning, and our use of these datasets resulted in similar and consistent evaluation results with the published studies.

With the above traces representing different but typical workload characteristics, this evaluation intends to answer, among other things, the following questions: Can the SiLo locality algorithm compensate for the probabilistic similarity detection that may miss detecting large amounts of duplicate data? How effective is the SiLo similarity algorithm under different workload conditions? How is SiLo compared with existing state-of-the-art deduplication approaches in key performance measures?

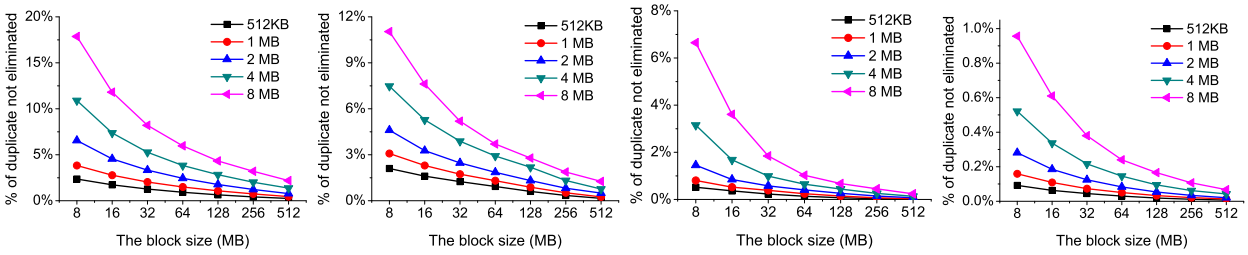
4.2 Interplay between Similarity and Locality

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.

From Figure 9 that shows the percentage of duplicate data not eliminated, we find 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 512KB), 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 512MB), the more locality of the backup stream will be retained and captured, allowing SiLo to eliminate more (i.e., 97%$\sim$99.9%) of redundant data regardless of the segment size.

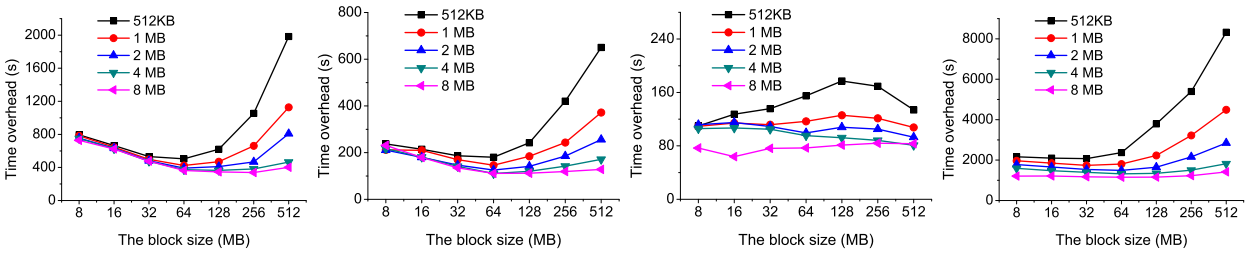
Although more redundant data can be eliminated by reducing the segment size or filling a block with more segmentsas indicated by the results shown in Figure 9, it results in more accesses to on-disks index and higher RAM usage due to the increased index entries in the SHTable (see Figure 5). As the deduplication-time-overhead results of Figure 10 clearly suggest, continuously decreasing the segment size or increasing the block size can become counterproductive after a certain point. From Figure 10, 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.

Figure 10 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 Figure 10 (c)). This may be explained by the fact that, with a very small block size (e.g., 8MB), there is little locality to be mined, resulting in frequent accesses to on-disk index. With a very large block size (e.g., 512MB), SiLo also runs slower because the increased disk accesses for locality exploitation may result in more unrelated segments being read in. The Linux-set are different from other datasets in Figure 10, because the average size of a Linux version is 110MB, which enables



(a) One-set (b) Inc-set (c) Linux-set (d) Full-set

Figure 9: Percentage of duplicate data eliminated as a function of block size and segment size.



(a) One-set (b) Inc-set (c) Linux-set (d) Full-set

Figure 10: Time overhead of SiLo deduplication as a function of block size and segment size.

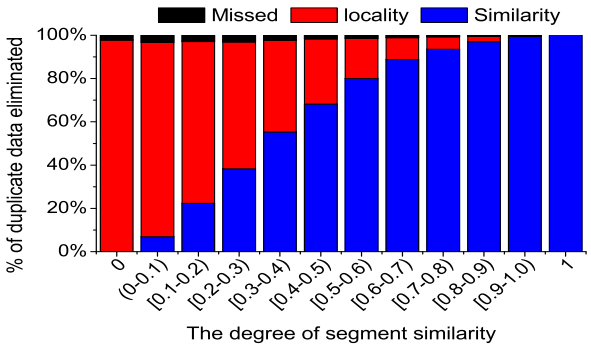


Figure 11: Percentage of duplicate data eliminated as a function of different similarity degrees on the Linux-dataset by the similarity-only approach and locality-only approach respectively.

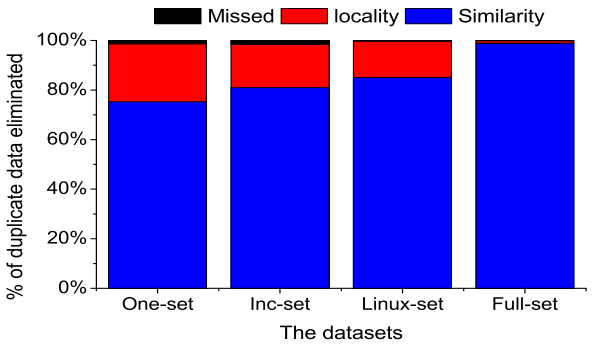


Figure 12: Percentage of duplicate data eliminated on four datasets by the similarity-only approach and locality-only approach respectively.

more related locality to be exploited at the block size of 256MB.

As analyzed above, there evidently exist an optimum segment size and an optimum block size, subject to a given workload and deduplication requirements (e.g., duplicate elimination or deduplication throughput). The choice of segment size and block size can be dynamically adjusted by the user's specific requirements (e.g., the backup throughput or duplicate elimination or the RAM usage).

Figures 11 and 12 suggest that the full exploitation of locality jointly with that of similarity can remove almost all redundant data missed by the similarity detection under all workloads. These results can be compared with Figure 2 and 3, then well verify our motivation of similarity and locality in Section 2. In fact, only an extremely small amount of duplicate data is missed by SiLo even on the datasets with weak locality and similarity.

4.3 Comparative Evaluation of SiLo

This subsection 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 four real-world traces described in Section 4.1 on these three systems. Note that in this evaluation SiLo assumes

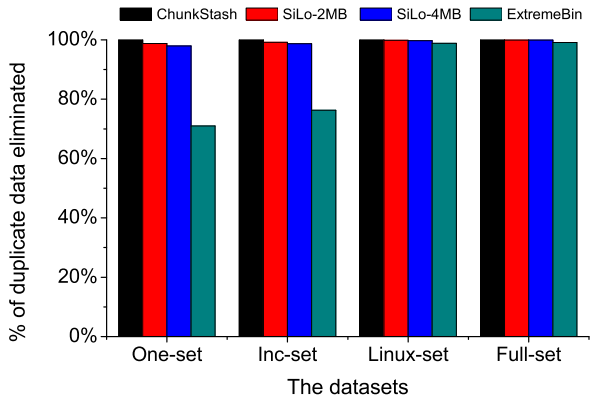


Figure 13: Comparison among ChunkStash, SiLo, and Extreme Binning in terms of percentage of duplicate data eliminated on the four datasets.

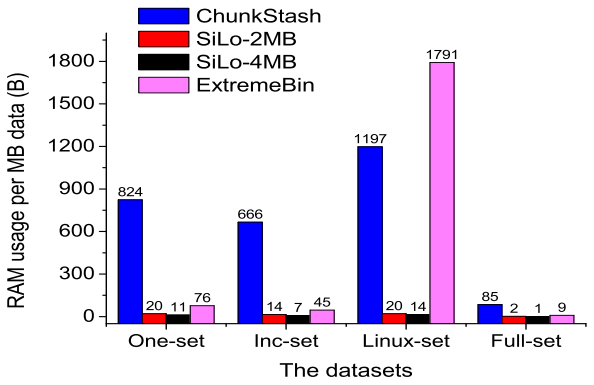


Figure 14: Comparison among ChunkStash, SiLo, and Extreme Binning in terms of RAM usage (B: RAM required per MB backup data).

a block size of 256MB, while SiLo-2MB and SiLo-4MB represent SiLo with a segment size of 2MB and 4MB respectively.

A. Duplicate elimination

Figure 13 shows the duplicate elimination performance of the three systems under the four workloads. Since ChunkStash does the exact deduplication, it eliminates 100% of duplicate data. Compared with Extreme Binning that eliminates 71%∼99% of duplicate data in the four datasets, SiLo removes about 98.5%∼99.9% of duplicate data. Note that, while Extreme Binning eliminates about 99% of duplicate data as expected in Linux-set and Full-set that has strong similarity and locality, it fails to detect almost 30% of duplicate data in One-set that has weak locality and similarity, and about 25% 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-2MB eliminates 99% of duplicate data even in One-set with both weak similarity and locality, and also removes almost 99.9% of duplicate data in Linux-set and Full-set 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 evaluated, achieving near-complete duplicate elimination (i.e., exact deduplication).

B. RAM usage

Figure 14 shows the RAM usage for deduplication among these three systems under the four 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-set that has 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 compact key signatures instead of full chunk-fingerprints, it still requires at least 6 bytes for each new chunk, resulting in a very large cuckoo hash table for millions of fingerprints. In addition, according to the open-source code of Cuckoo Hash [14], the ChunkStash system needs to allocate about two million hash table slots in advance to support one million index entries.

Since only the file similarity index needs to be stored in 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 the SiLo system stems from the interplay between its similarity algorithm, which groups many small correlated files into segments and extracts their similarity characteristics, and its locality algorithm, which groups contiguous segments of the backup stream into blocks to effectively exploit the locality residing in the backup-streams. On the other hand, the RAM usage for Extreme Binning depends on the average file size of the file set, 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.

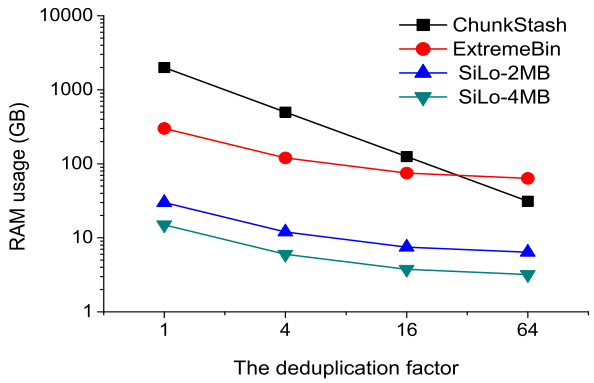


Figure 15: Comparison among ChunkStash, SiLo, and Extreme Binning in terms of RAM usage in PB-scale deduplication with different deduplication factors. We assume that the average file size is 200KB, and 80% of duplicate data are from duplicate files.

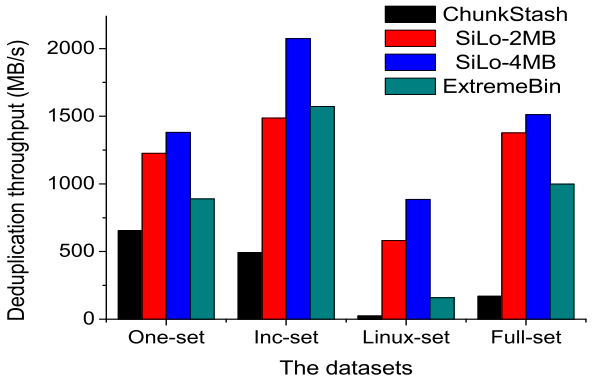


Figure 16: Comparison among ChunkStash, SiLo, and Extreme Binning in terms of deduplication throughput (MB/sec).

The RAM usage of the SiLo system remains relatively stable with the change in average file size in the four traces and is inversely proportional to the deduplication factor of the traces.

Now we analyze the RAM usage in a PB-scale deduplcation system for the three approaches. As a 2MB-segment needs 60 bytes of key index in the memory, SiLo takes up about 30GB of RAM in a PB-scale deduplication system. With 4MB-segments, SiLo's RAM usage is halved to 15 GB in a PB-scale deduplication system while its performance degrades gracefully as shown in Figures 9 and 10. Extreme Binning needs almost 300GB of RAM space with an average file size of 200KB while ChunkStash consumes almost 2TB of RAM space to maintain a global index in a PB-scale deduplication system.Figure 15 also shows RAM usage of these three approaches with different deduplication factors. According to [15], Sparse Indexing uses 170GB of RAM space for a PB-scale deduplication system, whereas it estimates that DDFS would require 360GB RAM to maintain a partial index depending on locality in backup streams.

C. Deduplication throughput

Figure 16 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 6KB (e.g., Linux-set) to 10KB (e.g., One-set).

ChunkStash achieves an average throughput of about 335MB/sec with a range of 24MB/sec$\sim$ 654MB/sec on the four datasets. 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 dataset, thus adversely affecting the throughput. Extreme Binning achieves an average throughput of 904MB/sec with a range of 158MB/sec$\sim$1571/sec on the four datasets, since it only needs to access the disk once per similar-file and eliminates the duplicate files in the memory. As SiLo-2MB makes at most one disk access per segment, it deduplicates data at an average throughput of 1167 MB/sec with a range of 581MB/sec$\sim$1486 MB /sec on the four datasets.

Although Extreme Binning runs faster than SiLo-2MB under Inc-set where many duplicate files exist, it runs much slower in other datasets. Since each bin stores all similar files and it tends to grow in size with the dataset size. 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 can help reduce disk accesses and increase the RAM utilization by preserving the locality layout in the memory.

Since SiLo uses significantly less RAM space than other approaches for a given dataset, SiLo can also boost the deduplication throughput by caching more index information in RAM to reduce accesses to on-disk index. In fact, the SiLo system can be dynamically configured with users' requirements such as the throughput and duplicate elimination by tuning the appropriate system parameters (e.g., the number of blocks in the cache, the segment size and block size, etc.). Therefore, compared with Extreme Binning and ChunkStash, SiLo is shown to provide robust and consistently good deduplication performance, achieving higher throughput and near-complete duplicate elimination at a much lower RAM overhead.

5 Related work

Data deduplication is an essential and critical component of backup/archiving storage systems. It not only reduces storage space requirements, but also improves the throughput of the backup/archiving systems by eliminating the network transmission of redundant data. We

briefly review the work that is most relevant to our SiLo system to put it in the appropriate perspective, as follows. LBFS [19] first proposes the content-based chunking algorithm with the adoption of the Rabin fingerprints [22], and applies it to the network file system to reduce transmission of redundant data. Venti [21] employs deduplication in an archival storage system and significantly reduces the storage space requirement. Policroniades etc. [20] compares the performance of several deduplication approaches, such as file-level, fixed-size chunking and content-based chunking.

In recent years, more attention has been paid to avoiding the fingerprint-lookup disk bottleneck and enabling more efficient and scalable deduplication in mass storage systems. DDFS [26] is the earliest research to propose the idea of exploiting the backup-stream locality to reduce accesses to on-disk index and avoid the disk bottleneck of inline deduplication. Sparse Indexing [15] also exploits the inherent backup-stream locality to solve the index-lookup bottleneck problem. Different from DDFS, Sparse Indexing is an approximate deduplication solution that samples index for fingerprint-lookup and only requires about half of the RAM usage of DDFS. But its duplicate elimination and throughput are heavily dependent on the sampling rate and chunks locality of backup streams.

ChunkStash [8] stores the chunk fingerprints on an SSD instead of an HDD to accelerate the index-lookup. It also preserves the backup-stream locality in the memory to increase the RAM utilization and reduce accesses to on-disk index. Cuckoo hash is used by ChunkStash to organize the fingerprint index in RAM, which is shown to be more efficient than Bloom filters in DDFS. ChunkStash study also shows that the disk-based Chunkstash scheme performs comparable to the flash-based ChunkStash scheme when there is sufficient locality in the data stream.

The aforementioned locality-based approaches would produce unacceptably poor performance of deduplication in the case of the data streams with little or no locality [3]. Several earlier studies [17, 5, 9, 4] propose to exploit similarity characteristics for small-scale deduplication of documents in the field of knowledge discovery and database. SDS [2] exploits the similarity of backup streams in mass deduplication systems. It divides a data stream into large 16MB blocks and constructs signatures to identify possibly similar blocks. A byte-by-byte comparison is conducted to eliminate duplicate data, which is also the first deduplication scheme that uses similarity matching. But the index structure in SDS appears to be proprietary and no details are provided in the reference paper. Extreme Binning [3] exploits the file similarity for deduplication to apply to non-traditional backup workloads with low-locality (e.g., incremental backup). It stores a similarity index of each new file in RAM and groups many similar files into bins that are stored on the disks, thus it eliminates duplicate files in RAM and duplicate chunks inside each bin by similarity detection.

SiLo is in part inspired by the Cumulus system and Bimodal algorithm. Cumulus is designed for file-system backup over the Internet under the assumption of a thin cloud [18]. 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 [14] 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 2-4 times larger than that of general algorithms, and finds more potential duplicate data among the boundaries of duplicate chunks.

Most recently, there have also been studies that explore the emerging applications of deduplication, such as the virtual machines [12, 7], the buffer cache [23], I/O deduplication [13] and flash [6, 11], suggesting an increasing popularity and importance of data deduplication.

6 Conclusion and future work

In this paper, we present SiLo, a similarity-locality based deduplication system that exploits both similarity and locality in backup streams to achieve higher throughput and near-complete duplicate elimination at a much lower RAM overhead than existing state-of-the-art approaches. SiLo exploits the similarity of backup streams by grouping small correlated files and segmenting large files to reduce the RAM usage for index-lookup. The backup-stream locality is mined in SiLo by grouping contiguous segments in backup streams to complement the similarity detection and alleviate the disk bottleneck due to 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, achieving a robust and consistently superior deduplication performance.

Results from experiments driven by real-world datasets show that the SiLo similarity algorithm significantly reduces the RAM usage while the SiLo locality algorithm helps eliminate most of the duplicate data that is missed by the similarity detection. And there exists a solution that optimizes the trade-off between duplicate elimination and throughput by appropriately tuning the locality and similarity parameters (i.e., the size of segment and block).

As our future work of SiLo, we plan to build a mathematical model to quantitatively analyze why SiLo works well with the combined and complementary exploitation

of similarity and locality and learn and adapt to the optimal parameter automatically by the real-time deduplication factor and other system status. Due to its low system overheads, we also plan to apply the SiLo system to other deduplication applications such as cloud storage or primary storage environments that desire to deduplicate redundant data with extremely low system overheads.

7 Acknowledgments

This work was supported by the National Basic Research 973 Program of China under Grant No.2011CB302301, the National High Technology Research and Development Program ("863" Program) of China under Grant No.2009AA01A401 and 2009AA01A402, NSFC No.60703046, 61025008, 60933002, 60873028, Changjiang innovative group of Education of China No.IRT0725, Fundamental Research Funds for the central universities, HUST, under grant 2010MS043, and the US NSF under Grants NSF-IIS-0916859, NSF-CCF-0937993 and NSF-CNS-1016609.The authors are also grateful to anonymous reviewers and our shepherd, Andy Tucker, for their feedback and guidance.

References

[1] AGRAWAL, N., BOLOSKY, W., DOUCEUR, J., AND LORCH, J. A five-year study of file-system metadata. *ACM Transactions on Storage (TOS) 3*, 3 (2007), 9.

[2] ARONOVICH, L., ASHER, R., BACHMAT, E., BITNER, H., HIRSCH, M., AND KLEIN, S. The design of a similarity based deduplication system. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference* (2009), ACM, pp. 1–14.

[3] BHAGWAT, D., ESHGHI, K., LONG, D., AND LILLIBRIDGE, M. Extreme binning: Scalable, parallel deduplication for chunk-based file backup. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009. MASCOTS'09. IEEE International Symposium on* (2009), IEEE, pp. 1–9.

[4] BHAGWAT, D., ESHGHI, K., AND MEHRA, P. Content-based document routing and index partitioning for scalable similarity-based searches in a large corpus. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining* (2007), ACM, pp. 105–112.

[5] BRODER, A. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings* (2002), IEEE, pp. 21–29.

[6] CHEN, F., LUO, T., AND ZHANG, X. CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives. In *FAST11: Proceedings of the 9th Conference on File and Storage Technologies* (2011), USENIX Association.

[7] CLEMENTS, A., AHMAD, I., VILAYANNUR, M., AND LI, J. Decentralized deduplication in SAN cluster file systems. In *Proceedings of the 2009 conference on USENIX Annual technical conference* (2009), USENIX Association, p. 8.

[8] DEBNATH, B., SENGUPTA, S., AND LI, J. 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.

[9] FORMAN, G., ESHGHI, K., AND CHIOCCHETTI, S. Finding similar files in large document repositories. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining* (2005), ACM, pp. 394–400.

[10] FULL-DATASET. http://en.amazingstore.org/xyj/.

[11] GUPTA, A., PISOLKAR, R., URGAONKAR, B., AND SIVASUBRAMANIAM, A. Leveraging Value Locality in Optimizing NAND Flash-based SSDs. In *FAST11: Proceedings of the 9th Conference on File and Storage Technologies* (2011), USENIX Association.

[12] JIN, K., AND MILLER, E. The effectiveness of deduplication on virtual machine disk images. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference* (2009), ACM, pp. 1–12.

[13] KOLLER, R., AND RANGASWAMI, R. I/O deduplication: utilizing content similarity to improve I/O performance. *ACM Transactions on Storage (TOS) 6*, 3 (2010), 1–26.

[14] KRUUS, E., UNGUREANU, C., AND DUBNICKI, C. Bimodal content defined chunking for backup streams. In *Proceedings of the 8th USENIX conference on File and storage technologies* (2010), USENIX Association, p. 18.

[15] LILLIBRIDGE, M., ESHGHI, K., BHAGWAT, D., DEOLALIKAR, V., TREZISE, G., AND CAMBLE, P. Sparse indexing: large scale, inline deduplication using sampling and locality. In *Proccedings of the 7th conference on File and storage technologies* (2009), USENIX Association, pp. 111–123.

[16] LINUX-DATASET. http://www.cn.kernel.org/pub/linux/kernel/.

[17] MANBER, U., ET AL. Finding similar files in a large file system. In *Proceedings of the USENIX winter 1994 technical conference* (1994), Citeseer, pp. 1–10.

[18] MICHAEL, V., STEFAN, S., AND GEOFFREY, M. Cumulus: Filesystem backup to the cloud. In *Proceedings of 7th USENIX Conference on File and Storage Technologies* (2009).

[19] MUTHITACHAROEN, A., CHEN, B., AND MAZIERES, D. A low-bandwidth network file system. In *Proceedings of the eighteenth ACM symposium on Operating systems principles* (2001), ACM, pp. 174–187.

[20] POLICRONIADES, C., AND PRATT, I. Alternatives for detecting redundancy in storage systems data. In *Proceedings of the annual conference on USENIX Annual Technical Conference* (2004), USENIX Association, p. 6.

[21] QUINLAN, S., AND DORWARD, S. Venti: a new approach to archival storage. In *Proceedings of the FAST 2002 Conference on File and Storage Technologies* (2002), vol. 4.

[22] RABIN, M. *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[23] REN, J., AND YANG, Q. A New Buffer Cache Design Exploiting Both Temporal and Content Localities. In *2010 International Conference on Distributed Computing Systems* (2010), IEEE, pp. 273–282.

[24] TAN, Y., JIANG, H., FENG, D., TIAN, L., YAN, Z., AND ZHOU, G. SAM: A Semantic-Aware Multi-Tiered Source Deduplication Framework for Cloud Backup. In *2010 39th International Conference on Parallel Processing* (2010), IEEE, pp. 614–623.

[25] XING, Y., LI, Z., AND DAI, Y. PeerDedupe: Insights into the Peer-Assisted Sampling Deduplication. In *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on* (2010), IEEE, pp. 1–10.

[26] ZHU, B., LI, K., AND PATTERSON, H. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies* (2008), USENIX Association, pp. 1–14.