A High-performance Post-deduplication Delta Compression Scheme for Packed Datasets

Yucheng Zhang*, Hong Jiang†, Mengtian Shi‡, Chunzhi Wang‡¶, Nan Jiang§, Xinyun Wu‡
*School of Information Engineering, Nanchang University, Nanchang, China
†Department of Computer Science & Engineering, University of Texas at Arlington, Arlington, USA
‡School of Computer Science, Hubei University of Technology, Wuhan, China
§School of Information Engineering, East China Jiaotong University, Nanchang, China
¶Corresponding author: chunzhiwang@vip.163.com
Email:{zhangyc_hust@126.com, hong.jiang@uta.edu, mengtianshi_hbut@163.com, chunzhiwang@vip.163.com, jiangnan1018@acm.org, xinyun@hbut.edu.cn}

***Abstract*—Data deduplication has become a standard feature in most storage backup systems to reduce storage costs. In real-world deduplication-based backup products, small files are grouped into larger packed files prior to deduplication. For each file, the grouping entails a backup product inserting a metadata block immediately before the file contents. Since the contents of these metadata blocks vary with every backup, different backup streams of the packed files from the same or highly similar small files will contain chunks that are considered mostly unique by conventional deduplication. That is, most of the contents among these unique chunks in different backups are identical, except for metadata blocks. Delta compression is able to remove those redundancy but cannot be applied to backup storage because the extra I/Os required to retrieve the base chunks significantly decrease backup throughput. If there are many grouped small files in the backup datasets, some duplicate chunks, called persistent fragmented chunks (PFCs), may be rewritten repeatedly. We observe that PFCs are often surrounded by substantial unique chunks containing metadata blocks. In this paper, we propose a PFC-inspired delta compression scheme to efficiently perform delta compression for unique chunks surrounding identical PFCs.**

In the process of deduplication, containers holding previous copies of the chunks being considered for storage will be accessed for prefetching metadata to accelerate the detection of duplicates. The main idea behind our scheme is to identify containers holding PFCs and prefetch chunks in those containers by piggybacking on the reads for prefetching metadata when they are accessed during deduplication. Base chunks for delta compression are then detected from the prefetched chunks, thus eliminating extra I/Os for retrieving the base chunks. Experimental results show that PFC-inspired delta compression attains additional data reduction by about 2x on top of data deduplications and accelerates the restore speed by 8.6%-49.3%, while moderately sacrificing the backup throughput by 1.2%-11.9%.

***Index Terms*—delta compression, data deduplication, rewriting algorithm, backup storage**

I. Introduction

The amount of digital data in the world is growing explosively. A study from International Data Corporation (IDC) indicates that the total amount of digital data in the world would reach 175 ZB in 2025 [1]. As a result, data storage efficiency has become a challenging task in computer systems in the big data era. Recent work reveals the wide existence of a huge amount of redundancy in backup storage systems [2], [3]. Thus, data deduplication has become a standard functionality in most backup systems to prevent redundant data from being stored in storage devices [4], [5]. Generally, data deduplication schemes split each input backup stream into chunks and calculate a cryptographically secure hash signature (also called fingerprint) for each chunk. By comparing fingerprints, duplicate chunks can be identified and removed from storage. Unique chunks are aggregated into a container, a fixed-size (e.g., 4 MB) structure, which is written to the disk when it is fully packed [6], [7]. When a backup completes, a recipe recording the fingerprint sequence of the backup stream is generated for future data restoration. During the restore process, required chunks are accessed according to the order of their fingerprints in the recipe to reconstruct the original backup stream [5].

In deduplication-based backup systems, multiple backups share identical chunks, rendering the logically contiguous chunks in the subsequent backups physically scattered in different containers, which is known as *chunk fragmentation*. During the restore process, time-consuming disk accesses for required chunks are the performance bottleneck. Chunk fragmentation increases the number of disk accesses during the restore process, and thus decreases restore speed. To address this problem, researchers propose rewriting algorithms that store (rewrite) some duplicate but fragmented chunks along with unique chunks to trade deduplication efficiency for restore performance.

In real-world backup products, before backup, files are grouped into larger packed files to increase the locality of the backup stream [2], [4], [8]. The format of a packed file resembles UNIX **"tar"** file. For each file grouped in the packed file, a backup product inserts a metadata block immediately before the file contents. Hence, files' metadata blocks are scattered in the packed file. Some contents in the matadata

This work was supported by the National Natural Science Foundation of China No. 61772180, 61902116, 62172160, 62062034, and 61862045; the Natural Science Foundation of Hubei Province No.2020CFB798, the Key R & D plan of Hubei Province(2020BHB004, 2020BAB012); the US NSF CNS-2008835.

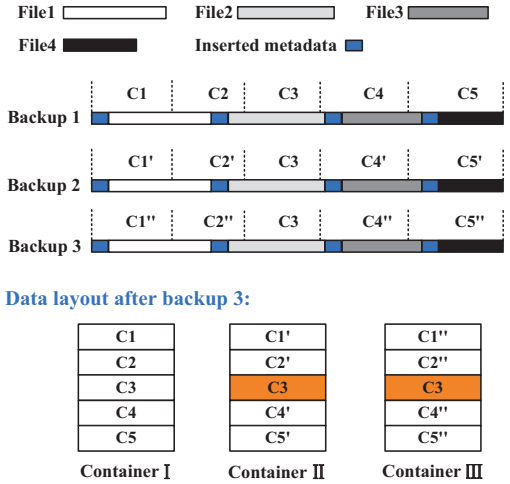


Fig. 1. Assuming that each backup contains 4 files that are grouped into a packed file and the 4 files in 3 backups are unchanged. After chunking, each backup is divided into 5 chunks. Since inserted metadata change in every backup, chunks $C1$, $C1'$ and $C1''$ respectively in three backups share most of contents. After backup 1, five chunks are stored in container I. When the system ingests backup 2, $C3$ is identified as a fragmented chunk and is rewritten since only one chunk (i.e., $C3$) in the container (i.e., container I) holding it is duplicate. After being rewritten, $C3$ is still stored along with unique chunks holding changed metadata in container II. Thus, when the system ingests backup 3, $C3$ is considered fragmented and rewritten again. Duplicate chunks that are repeatedly considered fragmented and rewritten are referred to as *persistent fragmented chunks* (PFCs), such as $C3$.

block, such as modification time, change every time when files are packed. As a result, during the deduplication process, chunks that contain files' metadata will be identified as unique chunks even though file contents in them are unchanged.

If there are too many small files in the backup dataset, the following phenomenon may appear for chunks in a backup stream: a small number of duplicate chunks are surrounded by a substantial number of unique chunks holding (unique) metadata. In the storage devices, these duplicate chunks would always be stored along with those unique chunks, even after being rewritten [9]. As a result, the aforementioned duplicates would repeatedly be identified as fragmented chunks and rewritten. We refer to such duplicate chunks as *persistent fragmented chunks (PFCs)*. Note that rewriting algorithms are designed to rewrite as few duplicates as possible. Thus, duplicate chunks stored in the containers that contain only a few duplicate chunks will be considered fragmented and rewritten. A simplified example is given in Fig. 1 to show how PFCs arise.

We observe that unique chunks in different backup streams surrounding identical PFCs are similar to each other since most of their contents are identical, expected for the metadata blocks. Taking chunks in Fig. 1 as an example. $C3$ is a PFC, chunks surrounding it in backup 2, i.e., $C1'$, $C2'$, $C4'$, and $C5'$, are respectively similar to corresponding chunks in backup 3, i.e., $C1''$, $C2''$, $C4''$ and $C5''$. Repeated strings among them cannot be removed by data deduplication, but can be eliminated by delta compression.

Delta compression is used to compress a data chunk relative to another. Given a target chunk B and a similar chunk A, delta compression encodes B relative to A and generates a *delta* file which consists of the contents existing in B but not in A. A is called the *base* of B. The delta file, which is expected to be much smaller than B, is then stored or transferred instead of B to obtain a space- or bandwidth-saving. When the chunk B is required, the delta file and its base, i.e., the chunk A, are retrieved and delta decoded to reconstruct it. Since it is orthogonal to data deduplication, delta compression can be used as a complementary technique for data deduplication. A main challenge facing the application of delta compression to the deduplication-based backup system is the I/O overheads required by reading base chunks, which decrease backup throughput to an unacceptable level [10], [11].

In some deduplication-based backup systems that preserve chunk locality in containers, such as EMC's backup products [6], [10], containers referenced by chunks of the on-going backup stream would be accessed for prefetching fingerprints to accelerate the process of duplicate detection by leveraging the locality. Here the locality refers to the fact that duplicate chunks in consecutive backup streams appear in the same order with a high probability. Since PFCs identified in a backup stream will appear as duplicate chunks in the following backup streams, the containers holding them will be accessed for prefetching fingerprints. During this process, if all chunks in these containers can be prefetched, i.e., piggybacked on the retrieval of the fingerprints, to serve as the potential base chunks, we can perform delta compression for unique chunks surrounding those PFCs, thus avoiding extra disk accesses for reading base chunks.

Motivated by above observations and analysis, we propose a PFC-inspired post-deduplication delta compression scheme, called PFC-delta, for deduplication-based backup systems. The main idea behind our scheme is to identify the containers holding PFCs and prefetch potential base chunks along with fingerprints when the system accesses these containers during the duplicate detection process. Then, PFC-delta detects similar chunks for unique chunks from the prefetched potential base chunks and performs delta compression for them if possible. Since base chunks are prefetched along with fingerprints, our scheme does not require extra disk accesses. Experimental results based on real-world datasets demonstrate that our scheme improves both compression ratio and restore performance on top of the traditional deduplication-based backup system, at the cost of a slight decrease of backup throughput.

The rest of the paper is organized as follows. In Section II, we present the background and related work. In Section III, we present our observations that motivate our work. In Section IV, we present the detailed design of our scheme. We present the experimental results in Section V and conclude the paper in Section VI.

II. Background and Motivations

A. Rewriting Algorithms

In a deduplication-based backup system that preserves the locality in containers, a restore cache is maintained in memory

during the restore process. To restore the original backup stream, referenced containers are loaded into the restore cache to provide requested chunks [7]. Restore performance is mainly decided by the number of disk reads for referenced containers. Fewer referenced containers equal fewer disk reads and higher restore performance. Chunk fragmentation significantly decreases restore speed since it increases the number of referenced containers. A chunk is considered *fragmented* if it is stored in a container that contains only a few duplicate chunks [12]. To improve restore performance, researchers propose rewriting algorithms that rewrite (duplicate) some fragmented chunks to remove references to the containers holding those fragmented chunks, so that the number of read-out containers during the restore process can be capped.

A container's *utilization* is defined as the fraction of its valid chunks, i.e., $\frac{the\ size\ of\ valid\ chunks}{the\ container\ size}$. Rewriting algorithms calculate utilizations of containers. Duplicate chunks stored in containers whose utilizations are smaller than a pre-defined threshold are considered fragmented. CBR [13], Capping [14], and HAR [12] are state-of-the-art rewriting algorithms. CBR and Capping buffer a portion of consecutive chunks in the on-going backup stream and calculate utilizations of containers using the information of buffered chunks. Instead of calculating containers' utilizations using the information of a portion of chunks during the backup process as in CBR and Capping, HAR calculates them using the information of all chunks when a backup completes.

Some schemes built upon the aforementioned three rewriting algorithms are also proposed. Based on the Capping design, Cao et al. [15] propose a container-flexible referenced-count scheme and a sliding look-back window to increase the accuracy of fragmented-chunks identifications. They also propose a combined chunk caching and forward assembly scheme to accelerate restore speed [16]. The scheme is able to adapt to different workloads with a fixed-size memory.

B. Inefficiency of Packed Datasets for Deduplication

Backup storage datasets are tied to the software that generates them, such as EMC NetWorker or Symantec NetBackup [17]. The software packs small files into large files and copies them to the storage system [2], [8]. Packed files are similar to UNIX "tar" files. Packing smaller files into large files increases the backup stream locality, contributing to higher caching efficiency for writing and thus higher write throughput.

tar (tape archives) was originally designed for storing files conveniently on magnetic tape, but has been widely used for decades and thus is difficult to be replaced for compatibility reasons. A study on data deduplication for HPC storage suggests that tar-type files account for about 38% of the total storage space [18]. Therefore, in this paper, we take tar-type files as an example to show the efficiency of our approach in improving compression benefits as well as the restore performance on datasets including such packed files.

A tar-type file consists of a sequence of entries, one per file, each containing a metadata block and a sequence of data blocks. The metadata block includes metadata for that file, including its name, path, size, ownership, modification time, etc. The contents of the file are stored in one or more data blocks. To avoid extra "seeks" during extracting a single file, data blocks are placed immediately after the metadata block. As a result, in a tar-type file, metadata blocks are interspersed with data blocks, as shown in Fig. 2. Data deduplication identifies redundant data at the granularity of chunks. Metadata in a metadata block, such as modification time, change with every backup, producing many unique chunks that adversely impact deduplication efficiency.

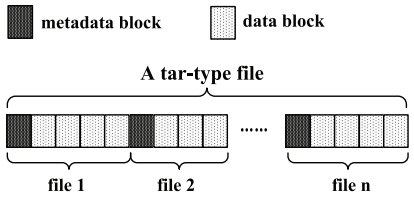


Fig. 2. An example of a tar-type file consisting of n files.

Assuming that the contents of files packed into larger files are unchanged in multiple consecutive backup streams which is normal for backup datasets, duplicate chunks will be interspersed with unique chunks. If the files packed in the tar-type files are small, a small number of duplicate chunks in the backup stream will be surrounded by a large number of unique chunks containing metadata. After data deduplication, these duplicate chunks will be stored in the containers with low utilization, even after being rewritten, and thus will be repeatedly identified as fragmented chunks and rewritten. We refer to such duplicate chunks as persistent fragmented chunks (PFCs). PFCs reduce the efficiency of rewriting algorithms, as well as decreasing restore performance. In some datasets, PFCs account for more than 80% of rewritten chunks [9].

To solve the problem, Lin et al. [17] propose *mtar* that transforms tar-type files into a more deduplication-friendly format by separating metadata blocks from data blocks [17]. However, mtar suffers from two major drawbacks. First, by physically separating a files data from its metadata, *mtar* changes file restoration I/Os from sequential to random and complicates both write and read processes. Second, it is only designed for tar-type files, thus needing to track changes to file format, a moving target. Changes to the file format cause the metadata blocks to change (e.g., length) and result in reduced deduplication efficiency, requiring engineers to address the change. Zhang et al. [9], [19] observe that the existence of PFCs is due to the common strategy of existing rewriting algorithms that stores rewritten chunks along with unique chunks to preserve the locality, rendering PFCs to be always stored in the containers with low utilizations. They propose DePFC that identifies and groups PFCs to increase the utilization of containers storing PFCs, making grouped PFCs no longer fragmented. However, DePFC fails to remove redundant data among similar chunks holding metadata blocks.

C. Post-deduplication Delta Compression

Recently, delta compression has gained increasing popularity due to its ability to eliminate redundant data among

similar chunks whose redundancy eludes the detection of data deduplication. Since it is orthogonal and complementary to data deduplication, delta compression can be used to compress post-deduplication chunks in storage and file-transfer systems. A study on workloads of EMC shows that delta compression attains additional compression benefits between 1.4x-3.5x on top of data deduplications [11]. Researchers in EMC suggest that I/O overheads required by the retrieval of base chunks decrease the backup throughput to an unacceptable level [10]. We also confirm this conclusion in our evaluation in Section V-D. As a result, delta compression cannot be applied to high-performance backup systems [10], [11]. Our proposed technique, PFC-inspired delta compression (PFC-delta), aims to reduce I/O overheads of reading base chunks to make post-deduplication delta compression feasible for backup storage.

PFC-delta is essentially a technique to prefetch potential base chunks with high accuracy. Thus, it is orthogonal to any similarity-detection and delta-encoding based approaches. The traditional post-deduplication delta compression approaches either try to detect similar chunks among all stored chunks such as TAPER [20], Neptune [21], and QuickSync [22], or detect similar chunks with strong locality such as SIDC [10]. In contrast, our scheme only considers and prefetches similar chunks surrounding identical PFCs as potential base chunks. Moreover, traditional approaches detect base chunks before reading them from the storage device, while our scheme prefetches potential base chunks before detecting them.

III. Observations and Motivations

Lots of real-world workloads contain substantial small files. In typical file systems, more than 80% of the total number of files are smaller than 64 KB [23], [24]. Moreover, in specific workloads such as source code and website snapshot, small files dominate the whole workloads. After ingested such datasets that have been preprocessed for larger packed files (Section 1), the backup storage will contain substantial unique chunks resulting from the interspersed unique metadata blocks that each describe an original small file. We observe that most of the contents of such chunks (i.e., the data portions of the pre-packing small files) among different backups are identical, except for the metadata portions. These identical contents cannot be removed by data deduplication but can be eliminated by delta compression. However, delta compression cannot be applied to high-performance backup systems due to the disk bottleneck introduced by reading the base chunks.

We also observe that the aforementioned similar chunks among different backup streams often surround identical PFCs. For deduplication-based backup systems adopting the strategy of grouping consecutive chunks into containers, such as EMC's backup products, containers holding duplicates will be accessed during the duplicate detection stage for prefetching fingerprints to accelerate the duplicate detection [6], [10]. A salient feature of PFCs is that if they appear in one backup stream they would also appear in the subsequent backup streams. This means that containers, which hold PFCs and would be accessed for prefetching fingerprints in the following backups, are predictable. This, combined with the fact that unique chunks surrounding identical PFCs contain substantial redundancy, motivates us to propose a PFC-inspired delta compression scheme, called PFC-delta. When the system accesses containers holding PFCs for prefetching fingerprints during the duplicate detection stage, PFC-delta prefetches chunks along with fingerprints to serve as the potential base chunks, thus eliminating extra disk accesses for base chunks.

IV. Design and Implementation of PFC-inspired Delta Compression

A. System Architecture

The goal of PFC-delta is to perform delta compression for unique chunks surrounding identical PFCs after data deduplication, without dramatically decreasing backup throughput. There are five basic data structures in the system, namely, container, segment, fingerprint cache, $PCID_{recent}$, and base chunk cache, which are detailed below:

- A *container* is a fixed-sized storage unit consisting of two sections: a data section containing chunks and deltas and a metadata section storing the descriptors for these chunks and deltas including fingerprints, super-features, and positions of the corresponding chunks and deltas in the container. Super-features are used to detect similar chunks [25], [26].
- A *segment* consists of a number (e.g., 2048) of consecutive chunks in the backup stream. The data reduction workflow of the system includes multiple stages, i.e., deduplication, rewriting, similarity detection, and delta compression, each requiring a buffer to enable pipelining of these stages for high backup throughput. Segments are processed one by one in each buffer.
- A *fingerprint cache* contains fingerprints in the accessed containers. It is used to accelerate duplicate detection by leveraging the locality preserved in containers. When a fingerprint is matched in the on-disk fingerprint index, fingerprints in the corresponding container is loaded into the fingerprint cache. The subsequent fingerprints are likely to be matched in the cache due to the locality. When an eviction occurs, based on an LRU policy, all fingerprints from a container are evicted as a group.
- The $PCID_{recent}$ is a list containing ids of containers housing PFCs identified in the last backup.
- A *base chunk cache* contains chunks and their super-features in the recently accessed containers that contain PFCs. During the similarity detection process, PFC-delta detects similar chunks from the chunks in the cache to serve as the base chunks for delta compression. When eviction occurs, based on the LRU policy, all chunks and their super-features from a container are evicted as a group.

For convenience, we use acronym FC for fragmented chunk. Fig. 3 illustrates the architecture of PFC-delta. In the system, a backup stream is chunked, fingerprinted, and then grouped into segments. Segments are processed in each buffer one by one.

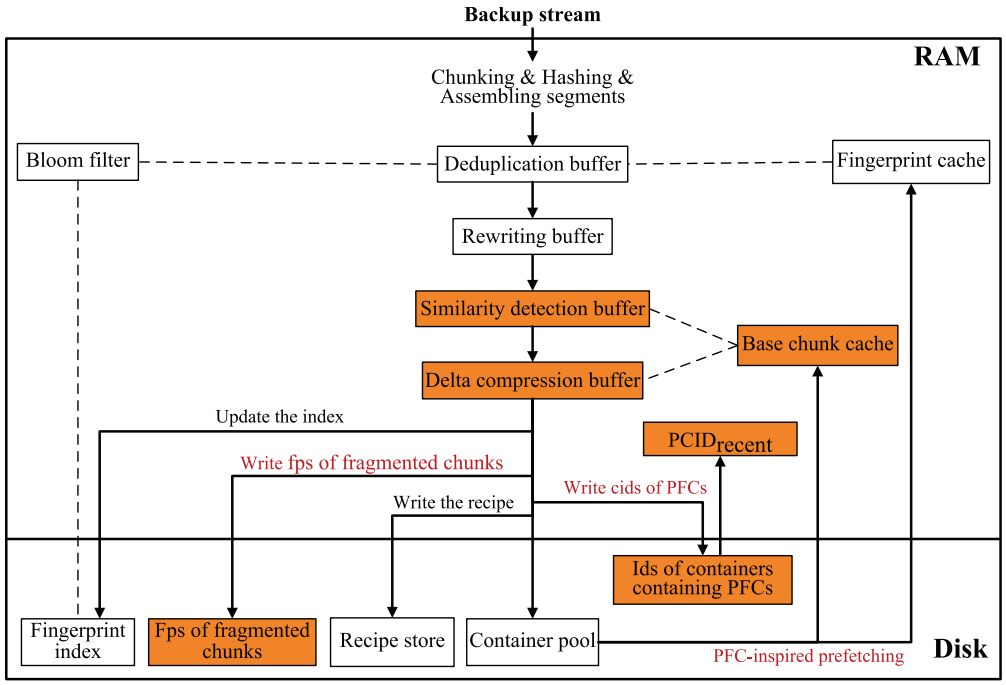


Fig. 3. The architecture of PFC-delta. Highlighted areas or words indicate newly added data structures or processes for PFC-delta.

In the deduplication buffer, duplicate chunks will be identified. Then, duplicate but fragmented chunks will be identified in the rewriting buffer. In the similarity detection buffer, the system computes super-features for unique chunks and FCs and detects their similar chunks. In the delta compression buffer, the system performs delta compression for unique chunks and FCs if their base chunks exist.

During the backup process, the system maintains one open container for each backup stream. After the input data are deduplicated and delta compressed, unique chunks, rewritten chunks, deltas, and their metadata are inserted to the currently open container. When fully packed, this container is sealed and written to the disk and a new (empty) container is opened for subsequent data. Meanwhile, files' recipes are stored in the disk for future restorations, and fingerprints of stored chunks and deltas are updated in the on-disk fingerprint index. Besides, fingerprints of FCs are written to the disk, which will be used for identifying PFCs in the subsequent backups.

B. PFC-inspired Base Prefetching

1) PFC Identification: Identified PFCs are likely to appear in the subsequent backups. Thus, they can be identified by checking the FCs in the on-going backup to see whether they have been rewritten in recent backups. If true, those FCs are PFCs. Existing rewriting algorithms vary in identifying FCs, leading to different patterns of PFCs being repeatedly identified as FCs. For CBR and Capping, PFCs identified in a backup would be identified as FCs in all the subsequent backups. Different from CBR and Capping, HAR identifies FCs in a backup and rewrites them in the next backup. Thus, for HAR, PFCs identified in a backup would appear in the third backup, namely, PFCs appear in every two backups.

When a backup completes, fingerprints of all rewritten chunks are written to the disk. At the beginning of a backup, fingerprints of rewritten chunks in the last backup (or the last but one backup if the rewriting algorithm is HAR) are loaded into the memory to construct an index, called FC_{recent}. When an FC is declared, the system further checks whether its fingerprint exists in FC_{recent}. If true, the FC is a PFC.

2) Base Chunk Prefetching: The process of our scheme to identify duplicates is the same as that of a typical deduplication-based backup system. When a chunk is presented for storage, its fingerprint is first compared against the fingerprint cache. If it exists in the cache (cache hit), the chunk is duplicate. Otherwise, the system checks a Bloom filter to determine whether the chunk is likely to exist in the on-disk fingerprint index. If so, the fingerprint index is checked, and the corresponding container is accessed for prefetching fingerprints it holds. Otherwise, the chunks is unique.

In order to support high-performance post-deduplication delta compression, before accessing a container for prefetching fingerprinits, our scheme checks whether this container contains PFCs identified in the last backup (or in the last but one backup if the rewriting algorithm is HAR). If so, the container may contain substantial potential similar chunks for the on-going backup stream, because those PFCs would appear in the on-going backup with a high probability and unique chunks surrounding them would be similar. Thus, chunks and super-features in the container should be prefetched along with fingerprints for both deduplication and delta compression. It is worth noting that super-features are stored in the metadata section of the container which has been introduced in Section IV-A.

Specifically, our scheme records ids of containers holding identified PFCs when a backup completes. At the beginning of

a backup, container ids recorded in the last backup (or the last but one backup if the rewriting algorithm is HAR) are loaded into memory to construct a structure, called $PCID_{recent}$. For a container to be accessed for prefetching fingerprints, $PCID_{recent}$ is checked. If id of this container exists in $PCID_{recent}$, fingerprints, super-features, and chunks in this container are read. Fingerprints are inserted into the fingerprint cache for deduplication, while chunks and super-features are inserted into the base chunk cache for similarity detection and delta compression. Otherwise, if the id of the container does not exist in $PCID_{recent}$, only fingerprints are read.

C. Memory Footprints

Compared with the traditional deduplication-based backup systems, our scheme introduces three data structures to support high-performance post-deduplication delta compression, namely, FC_{recent}, $PCID_{recent}$, and the base chunk cache. Assume that the minimal and expected chunk sizes are 2 KB and 8 KB, respectively, the same as LBFS [27]. Then the real average chunk size is about 10 KB. We also assume that the rewrite limit for the rewriting algorithm is 5%, suggested by CBR [13] and HAR [12]. Given a backup size of 20 GB, the FC_{recent} requires about 2 MB. The memory footprint required by $PCID_{recent}$ is negligible, because the id of a container only takes 8 bytes.

The size of the base chunk cache will be discussed in Section V-B. Note that our scheme is designed to compress unique chunks surrounding identical PFCs. For datasets without PFCs, the base chunk cache does not require extra RAM space because prefetching operations for base chunks would not be triggered.

V. Performance Evaluation

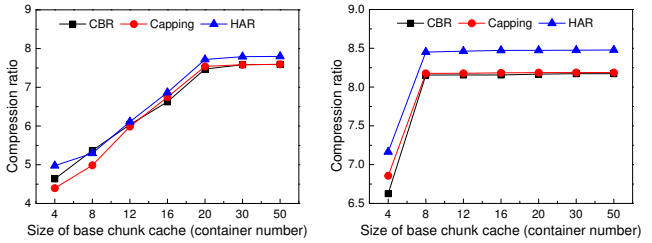
A. Evaluation Setup

Experimental Platform. We implement PFC-delta prototype in an open-source deduplication prototype system called Destor [28], on the Ubuntu 16.04.1 operating system running on an Intel Xeon W-2155 processor at 3.3 GHz. Destor only supports data deduplication, we modify its duplicate detection process to make it support PFC-inspired prefetching, and add two stages, i.e., similarity detection and delta compression, to make it support post-deduplication delta compression. The PFC-delta scheme is essentially a base-chunk-prefetching technique for packed datasets. It is thus orthogonal to any similarity detection and delta encoding approaches. We adopt Finesse [26] for similarity detection and Xdelta [29] for delta encoding.

System Configurations. In the prototype system, deduplication is configured with the Rabin-based chunking algorithm of which the minimal, average, and maximum chunk sizes are 2KB, 8KB, and 64KB respectively, the same as LBFS [27]. We index all fingerprints in the indexing stage, as used in EMC's backup products [6]. The container size is set to 4 MB. The settings of rewriting algorithms used in our evaluations, namely, CBR, Capping, and HAR, faithfully follow the recommended parameters of their original publications. For example, CBR's rewrite utility is set to 70%. For Capping, the capping level is set to 14. HAR's utilization threshold is set to 50%.

TABLE I
Workload characteristics of the two datasets used in the performance evaluation.

Name	Size	Workload descriptions
Webs	365 GB	133 days' snapshots of the website:news.sina.com [31], which are collected by *wget*.
Homes dirs	1817 GB	72 versions of software engineers' home directories containing source code, office documents, etc [9].



(a) Webs. (b) Homes dirs.

Fig. 4. Compression ratio of the deduplication-based backup system adopting PFC-delta as base container cache size varies on the two datasets.

For similarity detection, Finesse is configured to extract 12 features from each chunk, which are then grouped into 3 super-features for matching similar chunks. We use the LRU replacement scheme for the restore cache and set the restore cache size to 256 containers (i.e., 1 GB).

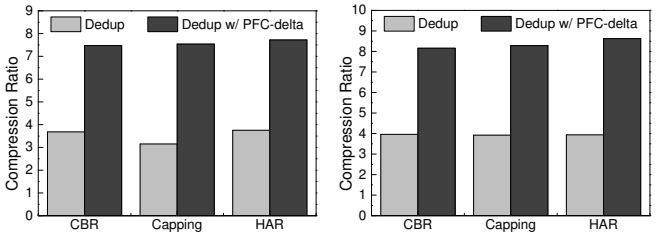
Performance Metrics. We use three metrics to evaluate the performance of our scheme. *Compression Ratio* is used to measure the total space-saving from data deduplication and delta compression, i.e., $\frac{the\ size\ of\ input\ data\ stream}{the\ size\ of\ stored\ data}$. *Speed Factor* [14], which is defined as the mean data size restored per container read, is used to measure the restore performance. Higher speed factors mean better restore performance. It is worth noting that each speed factor presented in our evaluations is the average of the last 20 backups, which is the same as [14] and [30].

Backup Throughput is measured by the throughput with which the input data stream is processed by both deduplication and delta compression. Backup throughput in our evaluations is tested on the last backup. We run each experiment six times to get the stable and average results.

Evaluated Datasets. Two datasets are used for evaluation as shown in Table I. These two datasets contain substantial small files, thus can be used to evaluate the efficiency of our scheme.

B. Compression Ratio

First, we study the sensitivity of the compression ratio to the base chunk cache size. For unique chunks surrounding PFCs in the on-going backup stream, the best candidates for delta compression are the corresponding unique chunks surrounding identical PFCs in the recent backup streams. Theoretically, a small-sized base chunk cache can detect substantial similar chunks containing identical PFCs because PFCs and the unique chunks surrounding them in consecutive backups appear in approximately the same order. However, a single backup



(a) Webs. (b) Homes dirs.

Fig. 5. Compression ratio of backup systems with and without PFC-delta, i.e., dedup and dedup w/ PFC-delta, for different rewriting schemes on the two datasets.

TABLE II
SPEED FACTORS OF DEDUPLICATION-BASED BACKUP SYSTEMS WITH AND WITHOUT PFC-DELTA, I.E. DEDUP AND DEDUP W/ PFC-DELTA, FOR DIFFERENT REWRITING SCHEMES ON THE TWO DATASETS.

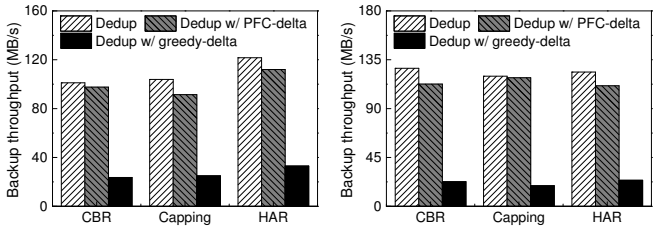
Webs	CBR	Dedup	1.57
		Dedup w/ PFC-delta	1.99 (+26.8%)
	Capping	Dedup	1.86
		Dedup w/ PFC-delta	2.02 (+8.6%)
	HAR	Dedup	1.38
		Dedup w/ PFC-delta	2.06 (+49.3%)
Homes dirs	CBR	Dedup	1.54
		Dedup w/ PFC-delta	2.03 (+31.8%)
	Capping	Dedup	1.65
		Dedup w/ PFC-delta	2.13 (+29.1%)
	HAR	Dedup	1.52
		Dedup w/ PFC-delta	2.07 (+36.2%)

may contain duplicate chunks itself. Such duplicate chunks are referred to as *self-referenced chunks*. Existence of self-referenced chunks may cause containers holding chunks of the on-going backup stream to be prefetched, instead of the containers stored in the recent backup streams, leading to a decrease number of detected base chunks. Increasing the size of the base chunk cache offers the opportunity to increase the number of detected base chunks. Fig. 4 presents the compression ratio of PFC-delta as the base container cache size increases on the two datasets. Results in the figure suggest that the compression ratios hit their maximum when the base chunk cache sizes are 20- and 8-container respectively. In what follows, the base chunk cache size is set to 20-container, i.e., 80 MB.

Next, we evaluate the impact of PFC-delta on compression ratio. Fig. 5 presents the compression ratio of deduplication-based backup systems with and without PFC-delta (i.e., *dedup* and *dedup w/ PFC-delta*) with three state-of-the-art rewriting algorithms, namely, CBR, Capping, and HAR, on the two datasets. Note that the compression ratio of dedup is achieved only by data deduplication. As shown in the figure, PFC-delta achieves 2× additional compression ratio on top of data deduplication. Specifically, on the Webs dataset, it improves the compression ratio of dedup with CBR, Capping, and HAR by 2.03×, 2.39×, and 2.06×, respectively. On the Homes dirs dataset, the improvements are 2.06×, 2.11×, and 2.19×, respectively.

C. Restore Performance

In this subsection, we evaluate the impact of PFC-delta on the restore performance. Table II indicates that our scheme



(a) Webs. (b) Homes dirs.

Fig. 6. Backup throughput of dedup, dedup w/ PFC-delta, and dedup w/ greedy-delta, with different rewriting schemes on the two datasets. Here dedup w/ greedy-delta represents a deduplication-based backup system greedily performing delta compression for unique chunks if their base chunks exist.

improves the restore performance of dedup by 8.6%-49.3%. Our scheme improves restore performance of dedup since it decreases the number of stored containers on top of data deduplication. Fewer stored containers indicates fewer disk reads for them during the restore process and thus greater speed factor. It should be noted that base chunks are also required during the restore process for decoding to reconstruct the original chunks, but our scheme does not require extra I/Os for reading them from the disk. Base chunks are stored in the containers selected for data deduplication, and those containers would be accessed even if our scheme is not applied. That is, when base chunks are required for decoding, they can be directly found in restore cache.

D. Backup Throughput

In this subsection, we evaluate the impact of our scheme on backup throughput by comparing the backup throughput of the deduplication-based backup systems with and without PFC-delta. In order to validate the impact of I/O overheads for reading base chunks in the traditional delta compression approach, we implement a deduplication-based backup system performing delta compression for all post-deduplication chunks of which base chunks exist, which we refer to as *dedup w/ greedy-delta*, and evaluate its backup throughput. As shown in Fig. 6, our scheme moderately decreases the backup throughput because it introduces two extra data reduction stages, i.e., similarity detection and delta compression. Specifically, on the Webs dataset, the decreases are 3.5%, 11.9%, and 7.9% respectively with three rewriting algorithms, namely, CBR, Capping, and HAR. On the Homes dirs dataset, the decreases are 11.4%, 1.2%, and 10.2% respectively. Besides, dedup w/ greedy-delta achieves much less backup throughput than the other two approaches, since it requires too many time-consuming random disk I/Os for reading back base chunks.

In summary, PFC-delta improves the compression ratio of dedup by about 2×, accelerates its restore performance by 8.6%-49.3%, while moderately decreasing the backup throughput.

VI. CONCLUSION

In deduplication-based backup storage, there are substantial redundant data among unique chunks of packed files, where the data blocks rarely change but metadata blocks change

with every backup. We observe that such chunks in different backup streams surrounding identical persistent fragmented chunks (PFCs) are similar to one another. We propose PFC-delta, a high-performance post-deduplication delta compression scheme for unique chunks containing metadata blocks surrounding identical PFCs. In the duplicate detection process, containers holding previous copies of the chunks being considered for storage will be accessed by prefetching fingerprints. Our scheme determines whether the containers to be accessed contain PFCs and prefetch potential base chunks along with fingerprints to avoid extra I/Os for reading the base chunks. Experimental results based on real-world datasets demonstrate that our scheme improves both compression ratio and restore performance of a typical deduplication-based backup system, without unacceptable backup throughput penalty.

REFERENCES

[1] "Data Age 2025," https://www.seagate.com/our-story/data-age-2025/, November 2018, The Digiization Universe of the World Seagate US.

[2] G. Wallace, F. Douglis, H. Qian, and et al, "Characteristics of backup workloads in production systems," in *the 10th USENIX Conference on File and Storage Technologies (FAST'12)*. San Jose, CA: USENIX Association, February 14 - 17 2012, pp. 1–14.

[3] G. Amvrosiadis and M. Bhadkamkar, "Identifying trends in enterprise data protection systems," in *the 2015 conference on USENIX Annual Technical Conference*. Santa Clara, CA: USENIX Association, July 08 - 10 2015, pp. 151–164.

[4] M. Lillibridge, K. Eshghi, D. Bhagwat, and et al, "Sparse indexing: Large scale, inline deduplication using sampling and locality." in *the 7th USENIX Conference on File and Storage Technologies (FAST'09)*, vol. 9. San Jose, CA: USENIX Association, February 24 - 27 2009, pp. 111–123.

[5] F. Guo and P. Efstathopoulos, "Building a high-performance deduplication system," in *the 2011 USENIX conference on USENIX Annual Technical Conference*. Portland, OR, USA: USENIX Association, June 15 - 17 2011, pp. 1–14.

[6] B. Zhu, K. Li, and R. H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system." in *the 6th USENIX Conference on File and Storage Technologies (FAST'08)*. San Jose, CA, USA: USENIX Association, February 26 - 29 2008, pp. 269–282.

[7] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, W. Xia, Y. Zhang, and Y. Tan, "Design tradeoffs for data deduplication performance in backup workloads." in *the 13th USENIX Conference on File and Storage Technologies (FAST'15)*, vol. 9. Santa Clara, CA, USA: USENIX Association, February 16 - 19 2015, pp. 331–345.

[8] W. Dong, F. Douglis, K. Li, R. H. Patterson, S. Reddy, and P. Shilane, "Tradeoffs in scalable data routing for deduplication clusters." in *the 9th USENIX Conference on File and Storage Technologies (FAST'11)*. San Jose, CA, USA: USENIX Association, February 15 - 17 2011, pp. 229–241.

[9] Y. Zhang, M. Fu, X. Wu, F. Wang, Q. Wang, C. Wang, X. Dong, and H. Han, "Improving restore performance of packed datasets in deduplication systems via reducing persistent fragmented chunks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 7, pp. 1651–1664, 2020.

[10] P. Shilane, M. Huang, G. Wallace, and et al, "WAN optimized replication of backup datasets using stream-informed delta compression," in *the Tenth USENIX Conference on File and Storage Technologies (FAST'12)*. San Jose, CA, USA: USENIX Association, February 14 - 17 2012, pp. 1–14.

[11] P. Shilane, G. Wallace, M. Huang, and W. Hsu, "Delta Compressed and Deduplicated Storage Using Stream-Informed Locality," in *the 4th USENIX conference on Hot Topics in Storage and File Systems*. Boston, MA, USA: USENIX Association, June 13 - 14 2012, pp. 201–214.

[12] M. Fu, D. Feng, Y. Hua, Z. Chen, Z. Chen, W. Xia, F. Huang, and Q. Liu, "Accelerating restore and garbage collection in deduplication-based backup systems via exploiting historical information," in *the 2014 USENIX conference on USENIX annual technical conference*. Philadelphia, PA, USA: USENIX Association, June 19 - 20 2014, pp. 181–192.

[13] M. Kaczmarczyk, M. Barczynski, W. Kilian, and C. Dubnicki, "Reducing impact of data fragmentation caused by in-line deduplication," in *the 5th Annual International Systems and Storage Conference (SYSTOR'12)*. Haifa, Israe: ACM Association, June 04 - 06 2012, pp. 1–12.

[14] M. Lillibridge, K. Eshghi, and D. Bhagwat, "Improving restore speed for backup systems that use inline chunk-based deduplication," in *the 11th USENIX Conference on File and Storage Technologies (FAST)*. San Jose, CA, USA: USENIX Association, February 12 - 15 2013, pp. 183–197.

[15] Z. Cao, S. Liu, F. Wu, G. Wang, B. Li, and D. H. Du, "Sliding look-back window assisted data chunk rewriting for improving deduplication restore performance," in *the 17th USENIX Conference on File and Storage Technologies (FAST'19)*. Boston, MA, USA: USENIX Association, February 25 - 28 2019, pp. 129–142.

[16] Z. Cao, H. Wen, F. Wu, and D. H. Du, "ALACC: Accelerating restore performance of data deduplication systems using adaptive look-ahead window assisted chunk caching," in *the 16th USENIX Conference on File and Storage Technologies (FAST'18)*. Oakland, CA, USA: USENIX Association, February 12 - 15 2018, pp. 309–324.

[17] X. Lin, F. Douglis, J. Li, X. Li, R. Ricci, S. Smaldone, and G. Wallace, "Metadata considered harmful to deduplication," in *7th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 15)*. Santa Clara, CA, USA: USENIX Association, July 6 - 7 2015.

[18] D. Meister, J. Kaiser, A. Brinkmann, and et al, "A Study on Data Deduplication in HPC Storage Systems," in *the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. Salt Lake City, Utah, USA: IEEE Computer Society Press, November 10- 16 2012, pp. 1–11.

[19] C. Zuo, F. Wang, P. Huang, Y. Hu, D. Feng, and Y. Zhang, "PFCG: Improving the restore performance of package datasets in deduplication systems," in *IEEE 36th International Conference on Computer Design (ICCD'18)*. IEEE, 2018, pp. 553–560.

[20] D. G. Korn and K.-P. Vo, "Engineering a differencing and compression data format." in *USENIX Annual Technical Conference, General Track*, 2002, pp. 219–228.

[21] Y. Hua, X. Liu, and D. Feng, "Neptune: Efficient remote communication services for cloud backups," in *INFOCOM, 2014 Proceedings IEEE*. Toronto, Canada: IEEE, April 27 - May 02, 2014, pp. 844–852.

[22] Y. Cui, Z. Lai, X. Wang, N. Dai, and C. Miao, "Quicksync: Improving synchronization efficiency for mobile cloud storage services," in *International Conference on Mobile Computing and NETWORKING*. Paris, France: ACM Association, September 07 - 11 2015, pp. 592–603.

[23] D. Meyer and W. Bolosky, "A study of practical deduplication," in *the 9th USENIX Conference on File and Storage Technologies (FAST'11)*. San Jose, CA, USA: USENIX Association, February 15 - 17 2011, pp. 229–241.

[24] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch, "A five-year study of file-system metadata," in *the 5th USENIX Conference on File and Storage Technologies (FAST'07)*. San Jose, CA, USA: USENIX Association, February 13 - 16 2007, pp. 31–45.

[25] A. Z. Broder, "Identifying and filtering near-duplicate documents," in *Combinatorial pattern matching*. Springer, 2000, pp. 1–10.

[26] Y. Zhang, W. Xia, D. Feng, H. Jiang, Y. Hua, and Q. Wang, "Finesse: fine-grained feature locality based fast resemblance detection for post-deduplication delta compression," in *the 17th USENIX Conference on File and Storage Technologies (FAST'19)*. Boston, MA, USA: USENIX Association, February 25 - 28 2019, pp. 121–128.

[27] A. Muthitacharoen, B. Chen, and D. Mazieres, "A Low-Bandwidth Network File System," in *the ACM Symposium on Operating Systems Principles (SOSP'01)*. Banff, Canada: ACM Association, October 21 - 24 2001, pp. 1–14.

[28] M. Fu, "Destor: An experimental platform for chunk-level data deduplication," https://github.com/fomy/destor, 2014.

[29] J. MacDonald, "File system support for delta compression," Ph.D. dissertation, Masters thesis. Department of Electrical Engineering and Computer Science, University of California at Berkeley, 2000.

[30] M. Fu, D. Feng, Y. Hua, X. He, Z. Chen, J. Liu, and et al, "Reducing fragmentation for in-line deduplication backup storage via exploiting backup history and cache knowledge," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, pp. 855–868, 2015.

[31] "Sina news," http://news.sina.com.cn/.