Applying Delta Compression to Packed Datasets for Efficient Data Reduction

Yucheng Zhang, Hong Jiang, *Fellow, IEEE*, Chunzhi Wang, *Member, IEEE*, Wei Huang, Meng Chen, Yongxuan Zhang, and Le Zhang

***Abstract*—Backup systems often adopt deduplication techniques for data reduction. Real-world backup products often group files into larger units (called packed files) before deduplicating them. The grouping entails inserting metadata immediately before the contents of each file in the packed file. Some metadata change with every backup, producing substantial similar (non-duplicate) chunks. Delta compression can remove redundancy among those similar chunks but cannot be applied to HDD-based backup storage because I/Os required for fetching base chunks result in severe throughput loss. For packed datasets, some duplicate chunks, called persistent fragmented chunks (PFCs), are rewritten every backup. We observe that corresponding chunk pairs surrounding identical PFCs are non-identical due to different metadata but similar to each other. In this article, we propose PFC-delta to perform high-performance delta compression for the aforementioned similar chunks on top of deduplication. PFC-delta identifies and prefetches potential base chunks stored along with PFCs by piggybacking on the routine I/Os during deduplication, thus avoiding extra I/Os. We also propose a hash-less delta encoding approach to reduce extra computational overheads. Evaluation results with four real-world datasets show that PFC-delta improves both compression ratio and restore performance, while increasing the backup throughput on all but one datasets.**

***Index Terms*—Delta compression, chunk fragmentation, backup system.**

Manuscript received 12 February 2023; revised 18 August 2023; accepted 17 September 2023. Date of publication 26 September 2023; date of current version 22 December 2023. This work was supported in part by the National Natural Science Foundation of China under Grants 62262042 and 62271239, in part by the U.S. NSF CNS-2008835, in part by the Jiangxi Provincial Natural Science Foundation under Grants 20224BAB202017 and 20224ABC03A01, in part by the Jiangxi Double Thousand Plan under Grant JXSQ2023201022, in part by the Key R&D plan of Hubei Province under Grant 2023BCB041, and in part by the Open Project Program of Wuhan National Laboratory for Optoelectronics under Grant 2021WNLOKF012. An early version of this paper was presented at the Proceedings of IEEE International Conference on Computer Design (ICCD), 2021 [DOI: 10.1109/ICCD53106.2021.00078]. Recommended for acceptance by A. Sivasubramaniam. *(Corresponding author: Wei Huang.)*

Yucheng Zhang, Wei Huang, Meng Chen, and Le Zhang are with the School of Mathematics and Computer Sciences, Nanchang University, Nanchang 330031, China (e-mail: zhangyc_hust@126.com; huangwei@ncu.edu.cn; chenmeng@ncu.edu.cn; zhangle@ncu.edu.cn).

Hong Jiang is with the Department of Computer Science and Engineering, The University of Texas at Arlington, Arlington, TX 76019 USA (e-mail: hong.jiang@uta.edu).

Chunzhi Wang is with the School of Computer Science, Hubei University of Technology, Wuhan 430068, China (e-mail: chunzhiwang@vip.163.com).

Yongxuan Zhang is with the School of Mathematics and Computer, Yuzhang Normal University, Nanchang 330088, China (e-mail: zyx126com@126.com).

Digital Object Identifier 10.1109/TC.2023.3318404

I. Introduction

DATA backup is an important approach to protect primary data. A salient feature of backup workloads is high redundancy among multiple backup streams [1], [2]. Thus, backup systems often use data deduplication to remove redundant data [3], [4]. Generally, data deduplication schemes divide each input backup stream into multiple chunks and calculate a fingerprint for each chunk. By indexing chunks' fingerprints, both duplicate and non-duplicate (unique) chunks can be identified. Deduplication-based backup systems do not store duplicate chunks for space-saving but refer them to their physical copies via small-size references. Unique chunks are aggregated into fixed-size (e.g., 4 MB) containers for storage [5], [6]. At the end of a backup, a *recipe* file that records the fingerprint sequence of the backup stream is stored for future data restoration.

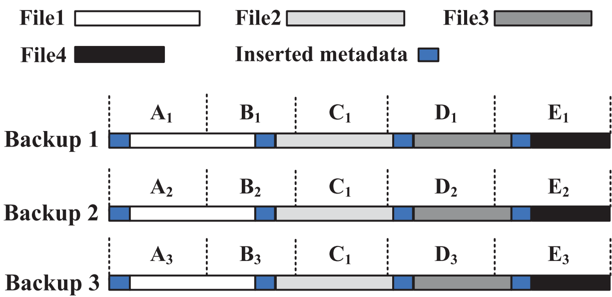
When files in the backup system are requested, chunks are accessed one by one to reassemble the original files [4], [7], [8]. The read unit during this (data restoration) process is a container. Since chunks of a backup stream are scattered across containers, also known as *fragmentation* [9], [10], the restore process requires a large number of disk accesses. For HDD-based backup systems, the performance bottleneck during data restoration is the disk I/Os for reading containers holding required chunks. To reduce disk I/Os during data restoration and accelerate the restore speed, researchers propose rewriting schemes that store (rewrite) some duplicates to reduce chunk fragmentation [10], [11], [12], [13].

To increase the backup stream locality, backup products often aggregate files to be backed up into larger units (called packed files in this article) before copying them to backup systems [1], [3], [14]. During the grouping process, each file to be backed up will be inserted some metadata of it immediately before the file contents, which is similar to the process of UNIX *"tar"*. Since some metadata such as modification time change every time the file being packed, chunks including those metadata will be considered non-duplicate (unique) even though file contents in them are unmodified. The impact of inserted metadata on deduplication efficiency can be significant because they are scattered in the packed file and thus can lead to a huge number of unique chunks.

If there are many small files in an input backup stream, a small number of duplicate chunks only containing unchanged files' contents may be surrounded by a large number of unique chunks including the changing metadata. If so, these duplicate

0018-9340 © 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on February 29,2024 at 21:56:46 UTC from IEEE Xplore. Restrictions apply.



Data layout after backup 3:

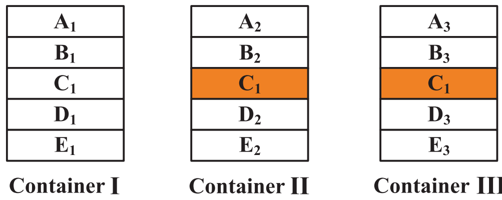


Fig. 1. Assuming that each backup stream is a packed file containing 4 files and the contents (excluding the inserted metadata) of the 4 files in the 3 backup streams are unchanged. After being chunked, each backup stream is divided into 5 chunks (4 chunks including metadata and 1 chunk containing only file contents). All chunks are considered unique and stored in container I after backup 1. During backup 2, C_1 is considered fragmented and rewritten since it references a container with low reuse (i.e., only a few chunks in this container are referenced by the current backup). After being rewritten, C_1 is still stored in a container with low reuse and thus is rewritten again during backup 3. C_1 is a *persistent fragmented chunk* (PFC).

chunks may be repeated rewritten in every backup, for two reasons. First, existing rewriting schemes are designed to rewrite as few duplicate chunks as possible. Thus, duplicate chunks in the containers with low reuse would be rewritten. Since duplicate chunks only containing unchanged files' contents are stored together with a large number of unique chunks surrounding them in the backup stream, they would be rewritten. Second, a rewritten chunk is often stored along with unique chunks adjacent to it in the backup stream to preserve the backup stream locality. Thus, those duplicate chunks would still be stored together with a large number of unique chunks including metadata after being rewritten. Duplicate chunks being repeatedly rewritten are referred to as *persistent fragmented chunks (PFCs)*. Fig. 1 gives an example to show how this problem arises.

For backup datasets containing many small files, we found that most of the corresponding chunks including inserted metadata in different backup versions surrounding identical PFCs are similar to each other. This is because they contain identical file contents but different metadata. Taking chunks in Fig. 1 as an example, A_2, B_2, D_2, and E_2, which surround a PFC (i.e., C_1), are respectively similar to the chunks that contain metadata and surround C_1, i.e., A_3, B_3, D_3, and E_3. Redundant data among such similar chunks cannot be eliminated by chunk-level deduplication, but can be removed by a similarity-based data reduction technique called *delta compression*.

Given a target chunk and a base chunk with similar content, the delta compression approach encodes the target chunk relative to the base chunk to generate a *delta* file consisting of their differences. The original target chunk can be reconstructed by decoding the delta file and the base chunk. Currently, the HDD is still often used as the storage device of backup systems due to its price-per-capacity superiority. Thus, this article focuses on HDD-based backup systems. Though delta compression can be used as a complementary technique to chunk-level deduplication for additional space-saving, it cannot be applied to HDD-based backup storage because extra disk accesses for reading back chunks from the storage device to serve as the base chunks result in severe backup throughput loss [15], [16].

Some backup products, such as DDFS [5], [15], access containers for prefetching metadata to speed up duplicate detection. During this process, if data chunks surrounding identical PFCs are also prefetched, or piggybacked, to serve as base chunks, we can perform delta compression for similar chunk pairs that surround identical PFCs without requiring extra I/Os. Except for I/O overheads, delta compression also introduces additional computational overheads for similarity detection and delta encoding. Traditional delta encoding approaches [17], [18] are time-consuming. This is because duplicate strings among the target and the base chunks may appear in different positions within the corresponding chunks and delta compression tools need to calculate fingerprints byte-by-byte on chunks and index these fingerprints to find alignments between the two chunks. Our study in Section III-B suggests that, for most of the similar chunks surrounding identical PFCs, their contents are identical, except for metadata. That is, their duplicate strings appear exactly in the same positions without any shift, which provides an opportunity to simplify the delta encoding process.

Motivated by above observations and analysis, we propose a PFC-inspired delta compression scheme, called PFC-delta, to perform high-performance delta compression for similar chunk pairs surrounding identical PFCs on top of chunk-level deduplication. PFC-delta builds on the deduplication strategy of grouping chunks into containers for storage and prefetching containers' metadata during deduplication. To eliminate disk I/Os for fetching base chunks, it identifies containers holding PFCs and prefetches potential base chunks in them by piggybacking on the I/Os for prefetching metadata during deduplication. To reduce computational overheads required by delta encoding, it adopts a hash-less delta encoding approach that is devoid of time-consuming fingerprinting and indexing operations as used in the traditional approaches for delta encoding similar chunk pairs whose duplicate strings appear exactly in the same positions. The contributions of this article are three folds:

- We propose a base chunk prefetching approach called *PFC-inspired prefetching* to eliminate extra disk I/Os for fetching base chunks. PFC-inspired prefetching identifies the containers holding PFCs and prefetches data chunks in them as potential base chunks, along with metadata when these containers are accessed for prefetching metadata during the deduplication process.
- We propose a delta encoding approach called *hash-less delta encoding* for similar chunk pairs whose duplicate strings appear exactly in the same positions to reduce computational overheads required by delta encoding.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on February 29,2024 at 21:56:46 UTC from IEEE Xplore. Restrictions apply.

Hash-less delta encoding replaces time-consuming fingerprinting and indexing operations as used in the traditional approaches with extremely fast byte-wise comparisons.
- Evaluation results with four backup datasets suggest that PFC-delta improves the compression ratio of the typical deduplication-based backup system by a factor of 1.3×–2.5× and accelerates the restore speed by 1.2×–2.06×, while increasing the backup throughput on all but one datasets.

The rest of the article is organized as follows. Section II presents the necessary background and related work. Section III presents the observations motivating our work. Section IV presents the detailed design of our scheme. Section V presents the experimental results and Section VI presents the conclusion of the article.

II. Background and Related Work

A. Backup and Restore Processes

Generally, deduplication-based backup systems store a chunk when it first appears for storage and refer to any subsequent chunks with duplicate contents of it identified by their secure hash values (fingerprints) by references instead of storing their data contents, thus saving space. A main challenge facing the deduplication-based backup storage is to organize the fingerprint index for detecting duplicates. Zhu et al. [5] group consecutive chunks of a backup stream into containers to preserve the chunk locality and use an on-disk full fingerprint index to trigger the locality-based caching to reduce disk I/Os. Some other researchers put the fingerprint index in memory and reduce its memory usage via sampling [3], [4], [19].

Data restoration is the reverse process of backup, where required chunks are retrieved one by one to reassemble the original data stream [6]. To reduce disk I/Os during the restore process, the I/O unit during this process is a container and the system keeps a restore cache in memory. That is, to access a chunk, the container holding this chunk is loaded into the restore cache, so that chunks that are stored in the same container and will be accessed in the near future can be read together without extra I/Os. For HDD-based backup systems, restore speed is mainly decided by the number of read-in containers during the restore process. Fewer read-in containers indicate higher restore speed.

B. Fragmentation and Rewriting Schemes

Due to deduplication, chunks of a backup stream are physically scattered in the storage devices, which is known as *fragmentation* [7], [9], [10], [11], [13]. Chunk fragmentation severely hurts the restore performance since it increases the number of read-in containers during data restoration, especially with the increasing number of backups. To solve the problem, rewriting schemes are proposed which store some duplicate chunks to remove references to some previously-stored containers, thus decreasing the I/O number during data restoration. Obviously, rewriting schemes trade deduplication ratio for restore performance. To minimize rewritten chunks as well as the decrease of deduplication ratio, the duplicate chunks selected for rewritten should be stored in the containers with low reuse [12]. Zou et al. [7] eliminate chunk fragmentation by using out-of-line reorganization operations after each backup.

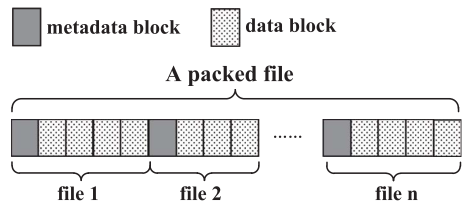


Fig. 2. The format of a packed file consisting of n files.

We define the *reuse ratio* of a container as the fraction of duplicate chunks it holds. Chunks referring to previously written chunks in the containers with low reuse ratios are considered fragmented and will be rewritten, thus avoiding those containers being read during data restoration. Rewriting schemes vary in calculating container's reuse ratio. Capping [11] and CBR [10] organize chunks of a backup into multiple segments, each of which consists of a sequence of chunks, and calculate containers' reuse ratios within each segment. HAR [12] calculates containers' reuse ratios using all chunks of a backup and uses them to identify fragmented chunks of the next backup.

Many optimizations to the aforementioned rewriting schemes are proposed. Instead of using a fixed capping level for all segments as in Capping, Cao et al. [20] propose a flexible container referenced count scheme to set different capping levels for different segments to increase the accuracy of fragmented-chunk identifications. Wu et al. [21], [22] observe redundancy among read-in containers resulting from rewriting schemes and propose selecting the containers with more distinct chunks for deduplication. Tan et al. [23], [24] group chunks into variable-size data groups, instead of the fixed-size one as used in the traditional backup storage, to identify fragmented chunks more accurately. Liu et al. [25] use an additional SSD to increase the number of chunks for Capping to calculate the reuse ratios of containers, which helps improve the accuracy of fragmented-chunk identifications.

C. Redundancy in Packed Datasets

Backup products often pack small files into larger units (called packed files) before copying them to the storage system [1], [14]. This strategy increases the stream locality which helps improve caching efficiency for backup. The format of a packed file is similar to that of a UNIX "tar" file, as shown in Fig. 2. Each file in a packed file consists of one metadata block and one or more data blocks. A file's metadata including its path, name, size, ownership, and modification time are placed in the metadata block, while the file's contents are stored in data blocks. A file's metadata block is placed together with its data blocks, so that, when this file is required, both its metadata and contents can be read together without requiring additional "seeks". Consequently, in a packed file, metadata blocks are interspersed with data blocks. Since some metadata such as modification time change every time the file is packed, chunks

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on February 29,2024 at 21:56:46 UTC from IEEE Xplore. Restrictions apply.

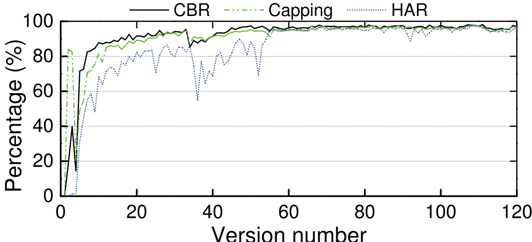


Fig. 3. Percentage of PFCs in all rewritten chunks for the state-of-the-art rewriting schemes (CBR, Capping, and HAR) for the WEB dataset.

including metadata blocks would be "unique" during deduplication even though the files' contents they contain are unmodified, leading to a decrease in deduplication efficiency.

If all files in a packed file are unmodified in multiple consecutive backup versions, duplicate chunks excluding metadata blocks would be interspersed with unique chunks including metadata blocks. If the packed files contain many small files, the following phenomenon may appear: a small number of duplicates would be repeatedly identified as fragmented chunks and rewritten in every backup because they would always be surrounded by a large number of unique chunks including metadata blocks when writing to the containers, even after being rewritten, which results in the low reuse ratios of the containers holding them. Such duplicate chunks, i.e., chunks being repeatedly rewritten, are referred to as *persistent fragmented chunks* (PFCs) [26].

Rewriting schemes often set a rewrite limit (e.g., 5% of chunks seen so far) to avoid too much rewrite. If the size of fragmented chunks exceeds this rewrite limit, restore performance would be decreased as some fragmented chunks will not be rewritten. Otherwise, deduplication efficiency would be decreased because PFCs would be rewritten in every backup [26]. In some packed datasets, PFCs may account for the vast majority (more than 90%) of rewritten chunks, the WEB dataset (see Section V-A) is a case in point, as shown in Fig. 3. In this case, PFCs can significantly impact either deduplication efficiency or restore performance.

Numerous approaches are proposed to avoid the impact of inserted metadata on the deduplication efficiency and restore performance. Instead of putting metadata immediately before data blocks, Lin et al. [27] colocate all metadata blocks at the end of the packed file to make metadata blocks localized and separated from data blocks. This approach changes file restoration I/Os from sequential to random and complicates both write and read processes. Zhang et al. [26], [28] group identified PFCs to increase the reuse ratios of the containers holding them and make them no longer fragmented. This approach improves restore performance but fails to eliminate redundancy among chunks with identical data blocks but different metadata blocks.

D. Delta Compression

Similarity Detection. Existing similarity detection approaches compute several features for each chunk for matching its similar chunk. The super-feature approach [29] computes a hash function for all consecutive strings with fixed size (e.g., 32 bytes) and selects the maximum (or minimum) hash as a feature. It extracts multiple features from each chunk and groups them into several super-features. Two chunks having a single super-feature in common are considered similar to each other. Zhang et al. [30] accelerate the process of feature calculation by exploiting the fine-grained locality among similar chunks. Zou et al. [31] speed up the feature computation process of the typical similarity detection approach by replacing the Rabin hash with a fast Gear hash combined with two optimizations to further reduce computational overheads.

Delta Compression. Delta compression can eliminate redundancy among chunks with similar contents whose redundancy eludes the detection of chunk-level deduplication [15], [16], [32], [33]. A typical delta compression tool called Xdelta [17] uses a byte-wise sliding window to identify repeated strings between the target and base chunks for differences calculation, which is very time-consuming. Xia et al. [34] propose a fast delta-encoding approach called Edelta, which replaces some of the time-consuming fingerprinting and indexing as used in Xdelta with fast byte-wise comparison to speed up the throughput of delta encoding.

Since delta compression is orthogonal and complementary to chunk-level deduplication, it can be used to compress post-deduplication chunks. Results from real-world product suggest that delta compression further improves the compression ratio of data deduplication by a factor of 1.4×–3.5× [16]. However, delta compression is not suitable for HDD-based backup systems because I/O overheads for fetching base chunks from the HDD severely decrease backup throughput [15], [16].

Zou et al. [8] propose to perform delta compression for unique chunks whose base chunks can be detected from the chunks of the last and the current backups. Their approach assumes that, after each backup, all chunks and the base chunks of delta-compressed chunks of the current backup can be reorganized to a compact data layout without chunk fragmentation. This approach, however, has a significant impact on the backup system because the backup system must perform the service-disruptive reorganizations frequently and finish them before the corresponding users' next backups. In this article, we focus on adding post-deduplication delta compression seamlessly to backup systems in a non-intrusive manner.

III. Observations and Motivations

A. PFC-Inspired Prefetching

Similar Chunks Surrounding Identical PFCs. We observe that, in a deduplication-based backup system, most of the corresponding chunks in different backup versions surrounding the same PFC are similar to each other (i.e., contain the same data blocks but different metadata blocks). In other words, an identified PFC in the current backup indicates multiple chunks surrounding the same PFC in the previous backup streams that are similar to the corresponding chunks in the current backup. Each pair of those similar chunks have identical data blocks but different metadata blocks. As a result, the corresponding chunks in the previous backup streams can serve as the base chunks for delta-compressing the chunks of the current backup.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on February 29,2024 at 21:56:46 UTC from IEEE Xplore. Restrictions apply.

Base Chunk Prefetching. In general, the basic I/O unit for deduplication-based backup systems is a container which consists of a data section and a metadata section. The former contains chunks while the latter contains chunks' fingerprints and the offsets in the container. To accelerate duplicate detection, the system needs to access containers' metadata sections during the deduplication process. This provides an opportunity to prefetch potential base chunks for delta compression without requiring extra disk I/Os. Specifically, when containers holding PFCs are accessed for prefetching metadata during deduplication, chunks in these containers are also prefetched, or piggybacked, to serve as potential base chunks.

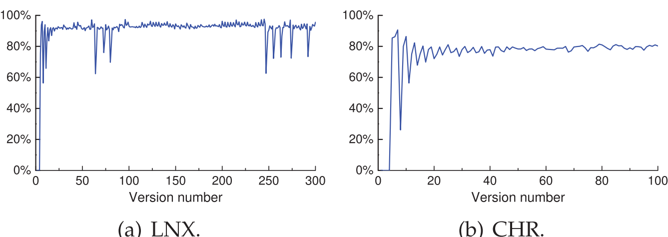
Taking chunks in Fig. 1 as an example. Assuming that C_1 is identified as a PFC in backup 2. When ingesting backup 3, the typical deduplication engine needs to prefetch the metadata in the metadata section of container II during deduplication. Since container II contains a PFC (i.e., C_1) that also appears in the ongoing backup, data chunks in this container are prefetched along with metadata. In this way, chunks A_2, B_2, D_2, and E_2 that can serve as the base chunks for delta-compressing the unique chunks A_3, B_3, D_3, and E_3 are prefetched without requiring extra I/Os.

Some schemes perform delta compression for all similar chunks in the system to minimize the redundant data [35], [36], [37]. Some other schemes detect similar chunks with strong locality for delta compression to reduce memory space for indexing the super-features [15], [16]. Our scheme aims to avoid extra I/Os introduced by delta compression by performing delta compression only for similar chunk pairs surrounding identical PFCs. Since some of the real-world backup datasets such as source code releases include a large amount of similar (non-duplicate) chunks introduced by changing metadata, performing delta compression for such datasets may significantly improve storage space efficiency.

B. Simplifying Delta Encoding Process

Existing delta encoding approaches use "COPY" and "INSERT" instructions to encode the input chunk relative to the base chunk [17], [18]. Duplicate strings shared by two similar chunks may appear in different positions in the two chunks. Before detecting common strings, delta encoding tools have to find alignments between the input and base chunks. Then the longest matches can be detected by expanding the contents of the aligned position byte-by-byte in both directions. To find the alignments, existing delta encoding approaches calculate fingerprints byte-by-byte on chunks, similar to content-defined chunking, and index fingerprints. However, calculating and indexing fingerprints are computationally expensive.

For similar chunk pairs surrounding identical PFCs, if their contents are identical except for the contents in the metadata blocks, their common strings appear exactly in the same positions within the corresponding chunks without any shift. This means that the time-consuming operations for calculating and indexing fingerprints are not necessary for delta encoding those similar chunks. For some packed datasets, similar chunk pairs whose redundancy appears exactly in the same positions account for a substantial proportion of all similar chunk pairs surrounding identical PFCs, such as the LNX and CHR datasets as shown in Fig. 4. Hence, a new delta encoding approach without the aforementioned time-consuming operations can significantly reduce computational overheads of delta encoding and has the potential to accelerate the backup throughput.



(a) LNX. (b) CHR.

Fig. 4. Percentage of similar chunk pairs whose redundancy appears exactly in the same positions within the corresponding chunks without any shift.

IV. Design and Implementation

A. System Architecture

PFC-delta is designed to remove redundant data between similar chunk pairs surrounding identical PFCs by performing high-performance delta compression. It builds upon a typical deduplication strategy of organizing consecutive unique chunks into containers for storage and prefetching containers' metadata during deduplication. To reduce I/O overheads required for fetching base chunks, PFC-delta identifies containers holding PFCs and prefetch potential similar chunks in them by piggybacking on the I/Os for prefetching metadata during deduplication, as detailed in Section IV-B. To reduce computational overheads required by delta encoding, we propose a hash-less delta encoding approach that is devoid of time-consuming fingerprinting and indexing operations as used in the traditional approaches, as detailed in Section IV-C.

Fig. 5 shows the data reduction workflow of a deduplication system with PFC-delta. A backup stream is firstly chunked and fingerprinted, and then processed by a deduplication engine to identify duplicates by indexing fingerprints. During deduplication, potential base chunks and their super-features are prefetched into the base chunk cache. The system then detects similar chunks from the base chunk cache for unique chunks and fragmented chunks and delta-encodes them if their similar chunks are found in the base chunk cache.

After being deduplicated and delta-compressed, unremoved chunks, deltas (delta-compressed chunks), and their metadata are written to a container. In our design, a container includes two sections: (1) a data section holding chunks as well as their super-features and deltas (delta-compressed chunks) and (2) a metadata section holding chunks' and deltas' fingerprints and offsets in the container.

B. PFC-Inspired Prefetching

In this subsection, we first introduce how to identify PFCs and then describe the strategy to prefetch potential base chunks in the containers holding PFCs.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on February 29,2024 at 21:56:46 UTC from IEEE Xplore. Restrictions apply.

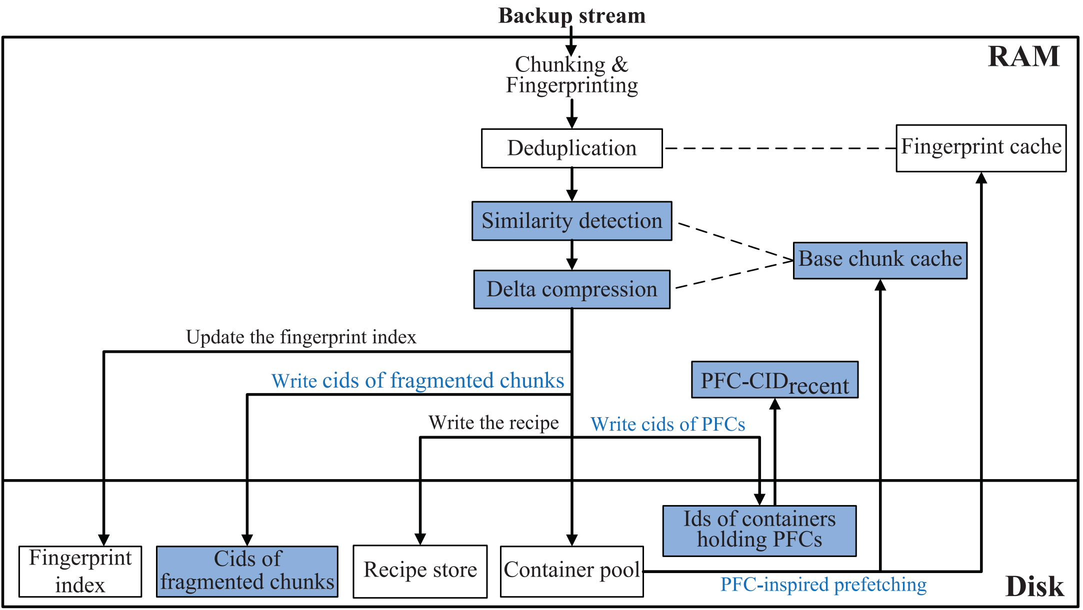


Fig. 5. The data reduction workflow of a deduplication system with PFC-delta. Newly added data structures and processes for supporting PFC-delta are shown in blue. Cid in the figure represents the container id which can be used to locate the physical position of the corresponding container.

1) PFC Identification: PFCs are duplicate chunks that are repeatedly identified as fragmented chunks. Existing rewriting schemes vary in how fragmented chunks are identified, leading to different patterns of PFCs being repeatedly identified as fragmented chunks. For CBR and Capping that identify fragmented chunks using the information of duplicates of the on-going backup, PFCs appear in every backup. Since HAR identifies fragmented chunks using the information of duplicates of the last backup, PFCs appear in every second backup.

A direct approach for identifying PFCs is to record and compare fingerprints of all fragmented chunks identified in the last and the current backups as used in [26]. However, this approach consumes too much memory because there might be a large number of fragmented chunks and each fingerprint takes 20 bytes. Recall that fragmented chunks are duplicate chunks in containers with low reuse ratios, which means that a container is the basic unit for identifying fragmented chunks. Hence, to reduce memory consumption, instead of recording fingerprints of fragmented chunks to identify PFCs, we record the ids of the containers with low reuse ratios. A container id is an 8-byte integer that can be used to locate the physical position of the corresponding container.

More specifically, when a backup completes, a file that records container ids of all identified fragmented chunks is written to the disk. At the beginning of the next backup (or the backup after next if the rewriting scheme is HAR), the aforementioned file is loaded and the container ids recorded in it are read to construct an in-memory lookup table, called $FC\text{-}CID_{last}$. When a fragmented chunk is declared, PFC-delta checks whether its container id exists in $FC\text{-}CID_{last}$. If so, this fragmented chunk is a PFC.

2) Base Chunk Prefetching:

Duplicate Detection and Removal (Deduplication). PFC-delta adopts the indexing (deduplication) scheme proposed by Zhu et al. [5], which puts the fingerprint index on the disk and uses a Bloom filter and a fingerprint cache to reduce disk accesses for checking the fingerprint index. For a new chunk, its fingerprint is first queried in the fingerprint cache. Upon a miss, the Bloom filter is checked to determine whether the data chunk is likely to exist in the system. If negative, the chunk is not a duplicate since Bloom filter does not return false negatives. Otherwise, the on-disk fingerprint index is checked. If the fingerprint matches in the fingerprint index, the container holding the physical copy of this chunk is accessed and the metadata in the metadata section are inserted into the fingerprint cache.

Base Chunk Prefetching. A key idea behind PFC-delta is to prefetch data chunks in the containers holding PFCs along with metadata when those containers are accessed during deduplication. It is easy to identify PFCs and the containers holding them. However, if we first identify PFCs in the current backup and then prefetch chunks in the containers holding those identified PFCs, only a few chunks would be prefetched to serve as the potential base chunks because the process to prefetch metadata precedes that to identify PFCs in the deduplication workflow. In other words, PFCs and the containers holding PFCs should be known before the process to prefetch metadata. To solve the problem, we prefetch chunks in the containers holding the PFCs that were identified in the last backup (or the last but one backup if the rewriting scheme is HAR). With this strategy, we can capture almost all potential similar chunk pairs surrounding identical PFCs because PFCs identified in a backup stream would also appear in the subsequent backup streams with a very high probability.

More specifically, when a backup completes, a file that records the ids of containers holding PFCs declared in this backup is stored. At the beginning of the next backup (or the backup after next if the rewriting scheme is HAR), the aforementioned file is loaded and container ids recorded in it

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on February 29,2024 at 21:56:46 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 Prefetching process of PFC-delta

Input: container ids of fragmented chunks in the last backup, *FC-CID*$_{last}$; ids of containers holding PFCs recorded in the recent backup, *PFC-CID*$_{last}$;

Output: container ids of emerging fragmented chunks, *FC-CID*$_{emerging}$; ids of containers holding PFCs recorded in the on-going backup, *PFC-CID*$_{emerging}$;

```
1:  Initialize a set, FC-CID_emerging;
2:  Initialize a set, PFC-CID_emerging;
3:  while the backup does not complete do
4:      Receive a chunk;
5:      if the chunk is duplicate then
6:          cid ← id of container referenced by the chunk;
7:          if cid exists in PFC-CID_last then
8:              Prefetch the whole container;
9:          else
10:             Prefetch metadata only;
11:             if the chunk is fragmented then
12:                 if cid exists in FC-CID_last then
13:                     Insert cid into PFC-CID_emerging;
14:                 else
15:                     Insert cid into FC-CID_emerging;
16:                 end if
17:             end if
18:         end if
19:     end if
20: end while
```

are read to construct an in-memory lookup table, called *PFC-CID*$_{last}$. When the deduplication engine accesses containers for prefetching metadata, it checks whether the id of the container to be accessed exists in *PFC-CID*$_{last}$. If true, chunks in the container and their super-features are fetched along with metadata and inserted into a base chunk cache to enable the subsequent post-deduplication delta compression. Otherwise, only the metadata in this container are fetched, the same as the traditional deduplication process [5]. Note that containers recorded in the aforementioned file generated by the last backup might have been reclaimed before the current backup starts and causes the PFC-inspired prefetching to be inefficient, which will be discussed in Section IV-D.

Algorithm 1 provides a more detailed description of the metadata and potential base chunk prefetching process of our design. We assume that the rewriting scheme is Capping [11], so a fragmented chunk is identified as a PFC if it is also a fragmented chunk in the last backup. Note that prefetching base chunks by piggybacking on the read operations for metadata eliminates extra I/Os for reading base chunks, but introduces extra transfer time for prefetching chunks. However, backup throughput would not be decreased because delta compression reduces disk I/Os for writing data.

Base Chunk Cache. Prefetched potential base chunks and their super-features are inserted into the base chunk cache. When eviction occurs, chunks and super-features from a container are evicted from the cache based on a Least Recently Used (LRU) policy. After deduplication, PFC-delta calculates super-features for chunks unremoved by deduplication and matching similar chunks from the base chunk cache. Matched similar chunks are then directly fetched from the base chunk cache to serve as base chunks for delta encoding.

C. Hash-Less Delta Encoding

Existing delta encoding tools have to calculate and index fingerprints to find alignments between similar chunks. However, for most of similar chunks surrounding identical PFCs, their duplicate strings appear exactly in the same positions without any shift, as detailed in Section III-B. Based on this observation, we propose a hash-less delta encoding approach for similar chunks surrounding identical PFCs without requiring time-consuming hashing and indexing operations. The hash-less delta encoding approach uses a byte-wise comparison at the beginning of the input and base chunks to detect the duplicate strings until the comparison fails. In practice, we accelerate the byte-wise comparing process by performing one 64-bit XOR operation for each pair of 8-byte strings at a time. When an XOR operation fails, a byte-wise comparison is used to obtain the longest match. The process to detect the longest un-matched strings is similar to that of the longest matched strings as detailed above.

It is obvious that the hash-less delta encoding approach is inefficient when the duplicate contents of the similar chunks are shifted, i.e., appear in different positions. Thus, we need to check whether the duplicate contents shared by two similar chunks are shifted before applying delta compression. Contents shifting often stems from the insertion or deletion operations. Since chunk boundaries are declared by content-defined chunking, e.g., Rabin-based content-defined chunking scheme, insertion and deletion operations often change chunk length. If the chunk lengths of two similar chunks are different, some duplicate contents between them have been shifted.

Even if the lengths of two similar chunks are the same, the contents may still have been shifted. To further check this, we sample several portions at the end of two chunks to see whether the contents of these portions in the two chunks are the same. The contents of a sampled portion of two similar chunks without contents shifting may still be different because the portion may be sampled from a metadata block. Thus, if the contents of one of the sampled portions of two similar chunks are common, duplicate strings in two chunks are considered appearing exactly in the same positions. We thus use the proposed hash-less delta encoding approach for delta encoding. Otherwise, we use Edelta [34].

D. Memory Footprints and Garbage Collection

Memory Footprints. PFC-delta introduces five data structures to support PFC-inspired base chunks prefetching and delta compression, namely, *FC-CID*$_{last}$, *PFC-CID*$_{last}$, *FC-CID*$_{emerging}$, *PFC-CID*$_{emerging}$, and a base chunk cache. The first four data structures record ids of containers with low reuse ratios and containers holding PFCs. The memory space consumed by them is negligible, for two reasons. First, the number of containers with low reuse ratios and the number of containers

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on February 29,2024 at 21:56:46 UTC from IEEE Xplore. Restrictions apply.

holding PFCs are limited. Second, a container id only takes 8 bytes.

Memory consumed by the base chunk cache will be evaluated and analyzed in Section V.B.3. Note that PFC-delta is designed to delta-compress similar chunk pairs surrounding the same PFCs. For datasets without PFCs, the base chunk cache would be empty and not require extra memory space.

Garbage Collection. Usually, each backup file would be specified a retention period. When its retention period expires, a backup file is deleted. Garbage Collection (GC) then sweeps (removes) invalid (unreferenced) chunks from the system by migrating valid chunks into new containers [4], [7], [12], [38], [39].

GC may cause the PFC-inspired prefetching to be inefficient because the containers holding fragmented chunks and PFCs declared in the last backup might have been reclaimed during GC. To solve the problem, we update the files that record the ids of containers holding fragmented chunks and PFCs generated by the last backup for all users after each GC. It should be noted that each backup stream has only two files to be updated. Compared with GC which involves a large number of I/Os, extra overheads for updating the files are negligible.

V. Performance Evaluation

A. Evaluation Setup

Experimental Platform. We build a PFC-delta prototype by extending an open-source deduplication system called Destor [40]. The prototype runs on a server machine with an Intel Xeon W-2155 CPU at 3.3 GHz and 64-GB RAM. Destor is a deduplication engine that only performs chunk-level deduplication, we modify its deduplication stage to enable PFC-inspired prefetching and add two extra processes after the deduplication process, i.e., similarity detection and delta compression, to support delta compression for similar chunk pairs surrounding identical PFCs.

System Configurations. All tested systems in this section are configured with the FastCDC [41] chunking scheme of which the minimum, average, and maximum chunk lengths are 2KB, 8KB, and 64KB respectively for chunking and sha1 [42] for fingerprinting. Three state-of-the-art rewriting schemes, i.e., CBR, Capping, and HAR, are used to identify fragmented chunks in our evaluations. The three rewriting schemes are configured with the default settings in their publications. We adopt Odess [31] to detect similar chunks, which is configured to compute 3 super-features for each chunk. The container size is set to 4 MB. During data restoration, the cache is configured as a 256-container-sized LRU cache, the same as [6].

Evaluated Datasets. Four real-world datasets containing substantial small files are used to evaluate the efficacy of PFC-delta. Each version of these datasets was packaged as a tar file. Characteristics of the four datasets are detailed in Table I.

Performance Metrics. Three key metrics are used to measure the performance of our scheme. *Compression Ratio*, defined as the size of the data stream before data reduction divided by the size of the data stream after data reduction,

TABLE I
Workload Characteristics of the Four Datasets Used in the Performance Evaluation

Name	Size	Workload descriptions
LNX	180 GB	300 versions of linux kernel [43] source code.
WEB	330 GB	120 days' snapshots of the website:news. sina.com [44].
CHR	284 GB	100 versions of source codes of chromium project.
HOME	1817 GB	72 versions of software engineers home directories including source code, office documents, etc.

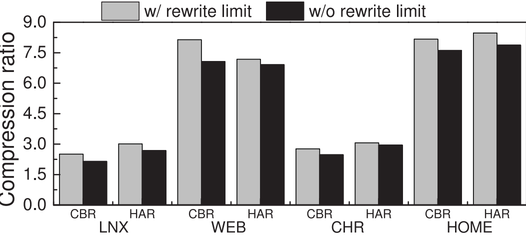


Fig. 6. Compression ratios of the deduplication-based backup system with PFC-delta with different rewriting schemes with and without rewrite limit (5%) for the four datasets.

is used to measure the total data reduction from the combination of data deduplication and delta compression. Note that for systems only adopting chunk-level deduplication in our evaluations for comparison, the compression ratio only reflects the data reduction from data deduplication, excluding delta compression. *Speed Factor* [11], defined as 1 divide by mean read-in containers per MB of restored data, i.e., $\frac{the\ size\ of\ data\ stream\ restored\ (MB)}{number\ of\ read-in\ containers\ during\ restore}$, reflects the restore performance. A higher speed factor indicates higher restore speed. In our evaluations, each speed factor presented is the average of the last 20 backups, which is the same as [11] and [45].

Backup Throughput is measured by the throughput with which the input backup stream is deduplicated and delta-compressed. The backup throughput presented in our evaluations is the average of the last 10 backups. For each experiment, we record the average results of five runs.

B. A Performance Study of PFC-Delta

In this section, we study the sensitivity of PFC-delta to rewrite limit, delta encoding approaches, and base chunk cache size.

1) Rewrite Limit: Rewriting schemes often have a rewrite limit (e.g., 5% of chunks), designed to avoid too many duplicates being rewritten [10], [45]. However, a smaller rewrite limit may lead to fewer identified PFCs which impact the efficiency of PFC-delta. This subsection evaluates the sensitivity of PFC-delta to rewrite limit. Dictated by its design principle, Capping does not require an extra rewrite limit. Thus, only CBR and HAR are evaluated in this subsection.

Figs. 6 and 7 suggest that rewriting schemes with a 5% rewrite limit achieve 3.8%–16.5% higher compression ratio and 6.4%–28.5% lower speed factors than that without the

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on February 29,2024 at 21:56:46 UTC from IEEE Xplore. Restrictions apply.

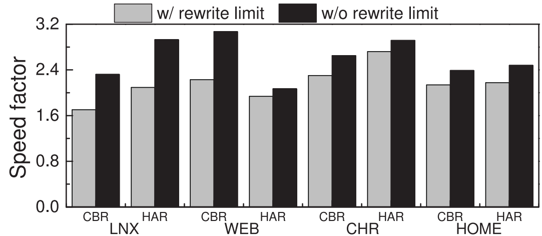
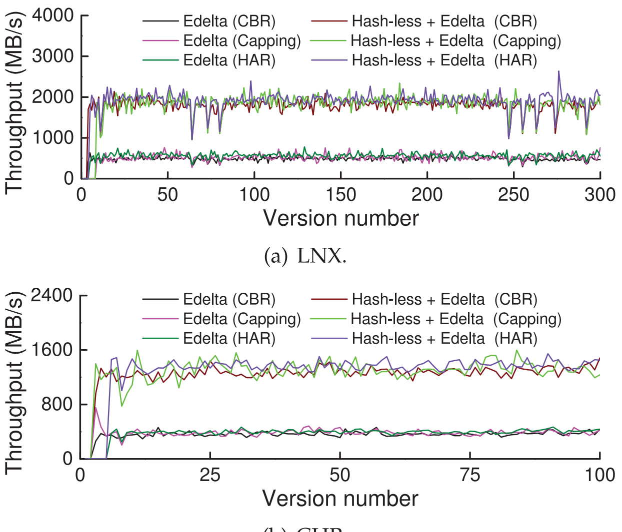


Fig. 7. Speed factors of the deduplication-based backup system with PFC-delta with different rewriting schemes with and without rewrite limit (5%) for the four datasets.



(a) LNX.

(b) CHR.

Fig. 8. Throughput of two delta encoding strategies for different rewriting schemes on the LNX and CHR datasets.

rewrite limit. In general, compression gains achieved by extra delta compression from removing the rewrite limit are less than that achieved by setting a rewrite limit which avoids excessive rewrites. Since the decrease in compression ratio due to removing the rewrite limit is often smaller than the improvement on the restore performance, in the remainder of the evaluation, we remove the rewrite limit for CBR and HAR.

2) Hash-Less Delta Encoding: Recall that PFC-delta adopts a delta encoding strategy called *Hash-less + Edelta* which combines *hash-less delta encoding* and *Edelta*. In this strategy, *hash-less delta encoding* is used for delta-encoding the similar chunk pairs whose duplicate strings appear exactly in the same positions without any shift, and *Edelta* is for the rest of similar chunk pairs. In this subsection, we test two versions of PFC-delta, with one adopting *Hash-less + Edelta* for delta encoding and the other using *Edelta* only. Fig. 8 compares the throughput of the two delta encoding strategies with different rewriting schemes on the LNX and CHR datasets. The results presented in Fig. 8 suggest that *Hash-less + Edelta* achieves much higher throughput than *Edelta*, by about 3×. This is because most similar chunk pairs are processed by hash-less delta encoding which is extremely fast and comparable to the main memory bandwidth.

We also evaluated the compression ratios of the two versions of PFC-delta and found that the system with *Hash-less + Edelta* achieves 0.19%–2.3% higher compression ratio than the one with *Edelta* (not shown because the improvements are limited). This is because of the fine-grained redundancy in the metadata blocks stemming from unmodified metadata. *Edelta* fails to detect some of such redundancy as it finds alignments between the two similar chunks by indexing coarse-grained sub-chunks, which *hash-less delta encoding* can successfully detect.

3) Base Chunk Cache Size: There can be multiple containers holding the same PFCs in the system because PFCs can be repeatedly rewritten. If multiple such containers are accessed during deduplication and prefetched into the base chunk cache, the total number of potential base chunks in the base chunk cache would be decreased because the similar candidates in the containers holding the same PFCs overlap. Thus, a larger base chunk cache may lead to more detected base chunks and more delta compression.

Fig. 9 depicts the compression ratios of PFC-delta with three rewriting schemes (i.e., CBR, Capping, and HAR) as the base chunk cache sizes varies for the four datasets. As shown in the figure, the compression ratios grow with the base chunk cache size and approach the maximum when the base chunk cache size is 16-container, i.e., 64 MB. Since 64 MB is affordable for a backup server, in the remainder of the evaluation, the base chunk cache size is set to 16-container.

4) Recording Fingerprints vs. Recording Container Ids: As described in Section IV.B.1, there are two methods to identify PFCs: recording fingerprints of fragmented chunks identified in each backup or recording ids of containers with low reuse ratios in each backup. This subsection discusses the memory footprint required by these two methods.

Typically, a fingerprint occupies 20 bytes, and a container id occupies 8 bytes. Table II indicates that recording fingerprints requires a memory footprint of up to 0.017% of the backup data size. When the backup data size is large, 0.017% can still be a significant value. Moreover, this is only the memory footprint required for processing a single backup data stream. In reality, the system may simultaneously handle hundreds of backup data streams. Overall, recording fingerprints requires a significant amount of memory.

On the other hand, if we record ids of the containers with low reuse ratios for identifying PFCs, the required memory footprint is only 0.00012% to 0.00017% of the backup data size, which is 1/113 to 1/70 of the memory required for recording fingerprints. Clearly, recording ids of containers with low reuse ratios can greatly reduce memory usage.

C. Comprehensive Evaluation of PFC-Delta

In this subsection, we comprehensively evaluate the performance of our scheme in terms of three key metrics: compression ratio, restore performance, and backup throughput. Since our scheme builds upon the typical deduplication strategy of organizing unique chunks into containers and prefetching containers' metadata during deduplication, a backup system adopting this deduplication strategy is implemented and used as the baseline for evaluating PFC-delta. To understand the impact of GC on PFC-delta's performance, we first evaluate

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on February 29,2024 at 21:56:46 UTC from IEEE Xplore. Restrictions apply.

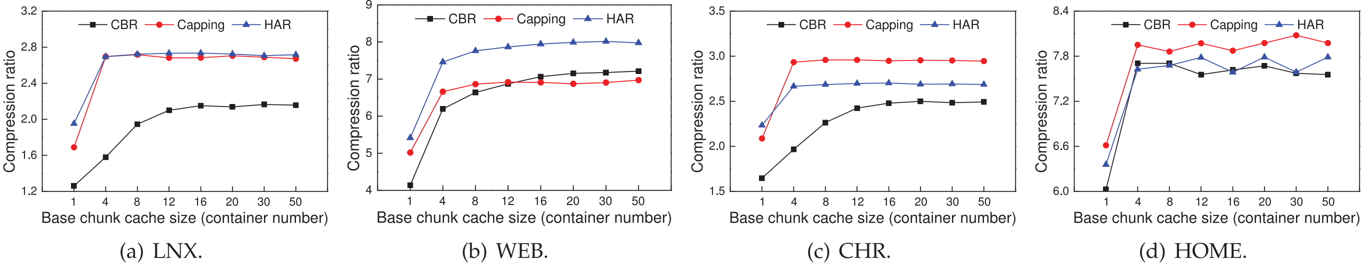


Fig. 9. Compression ratios of the deduplication-based backup system with PFC-delta and different rewriting schemes as base container cache size varies for the four datasets.

TABLE II
COMPARISON OF MEMORY FOOTPRINT AS A PERCENTAGE OF BACKUP DATA SIZE FOR TWO METHODS: RECORDING FINGERPRINTS AND RECORDING CONTAINER IDS USING THE CBR AND HAR REWRITING SCHEMES ON THE LNX AND CHR DATASETS. THE PRESENTED PERCENTAGES REPRESENT THE AVERAGE OF THE LAST 20 BACKUPS. CON-ID REPRESENTS CONTAINER ID

Dataset	CBR		HAR	
	Fingerprint	Con-id	Fingerprint	Con-id
LNX	0.012%	0.00017%	0.017%	0.00015%
CHR	0.011%	0.00014%	0.0095%	0.00012%

its performance without GC, then rerun the test with GC. In the test with GC, we retain only the latest 20 backups at any moment.

Compression Ratio. Fig. 10 on compression ratio indicates that, by performing delta compression for similar chunk pairs surrounding identical PFCs, our scheme achieves an additional 1.3×–2.5× compression beyond chunk-level deduplication. *PFC-delta* in the figure represents the deduplication-based backup system using PFC-delta for delta-compressing similar chunk pairs surrounding identical PFCs. In addition, GC slightly decreases compression ratio of PFC-delta by 0.1%–0.9%. This is because we update the file recording the ids of containers holding PFCs declared in the last backup after GC. GC decreases compression ratio because it lowers reuse ratios of containers referenced by the subsequent backups which leads to more rewritten chunks and thus lower compression ratios. The decreased compression ratio due to GC is marginal because we perform delta compression for rewritten chunks and most of rewritten chunks are delta-compressed.

Speed Factor. Fig. 11 on speed factor suggests that our scheme improves the speed factor of *Baseline* by 1.2×–2.06×. As stated in Section II-A, the speed factor is mainly decided by the number of read-in containers during data restoration and fewer read-in containers means a higher speed factor. PFC-delta improves the speed factor of *Baseline* for two reasons. First, base chunks prefetched by PFC-inspired prefetching do not require extra disk I/Os during restore because they are stored along with duplicate chunks which will trigger the prefetching operations in the first place during restore. Second, delta compression decreases the number of stored containers during backup as well as that of read-in containers during restore. Finally, GC has limited impact on speed factor.

Backup Throughput. In order to confirm the significant impact of additional I/Os introduced by delta compression for reading back base chunks from storage on the backup throughput, we implement a backup system, called *Greedy*, that performs delta compression on top of deduplication for all non-duplicate chunks whose base chunks exist. Moreover, to characterize the benefits of our proposed hash-less delta encoding approach, we also implement a version of PFC-delta that adopts *Edelta* [34] as the delta encoding tool, called *PFC-Edelta*, for backup throughput comparison. Although PFC-delta introduces extra overheads, it can accelerate the backup throughput because it decreases the I/O overheads for writing data to the disk.

Fig. 12 on backup throughput suggests that, on the LNX, CHR, and HOME datasets, *PFC-delta* accelerates the backup throughput of *Baseline* by 1.09×–1.43×. This is because, when the system processes these three datasets, writing data to the disk is the performance bottleneck and our scheme alleviates this bottleneck. On the WEB dataset, our scheme slightly decreases the backup throughput by 4.2%–5.1%. This is because the positive impact on backup throughput due to decreasing I/O overheads for writing data to the disk and the negative impact on that due to extra computational overheads required for similarity detection and delta encoding tend to cancel each other.

Fig. 12 also suggests that *PFC-delta* achieves 5.1%–10.6% higher backup throughput than *PFC-Edelta*. This is because the hash-less delta encoding requires much less computational overheads than Edelta. Even though the hash-less delta encoding is extremely fast, the improvements on backup throughput on some datasets (e.g., the WEB dataset) are insignificant because the performance bottleneck of the data reduction workflow has been shifted to other stages. In addition, *Greedy* achieves much lower throughput than the other three systems, which is consistent with the conclusions in [15], [16]. Finally, GC slightly decreases the backup throughput because GC may cause potential similar chunks in a container to be stored in two containers and thus leads to more containers being prefetched to supply potential base chunks.

In summary, by performing delta compression for chunks with identical data blocks but different metadata blocks surrounding identical PFCs without requiring extra I/Os, PFC-delta improves the compression ratio of the typical deduplication-based backup system by a factor of 1.3×–2.5×, accelerates the restore speed by 1.2×–2.06×, and increases the backup throughput on all but one datasets.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on February 29,2024 at 21:56:46 UTC from IEEE Xplore. Restrictions apply.

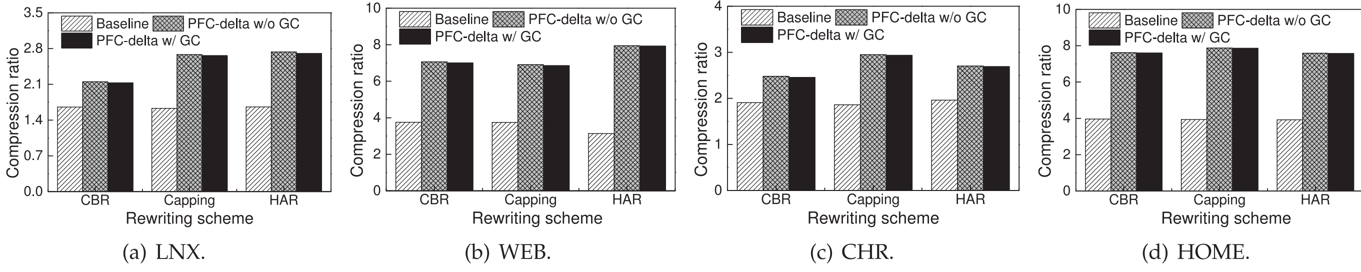


Fig. 10. Compression ratios of *Baseline* and *PFC-delta* without and with GC for different rewriting schemes for the four datasets. *PFC-delta* represents the deduplication-based backup system adopting PFC-delta for delta-compressing similar chunk pairs surrounding identical PFCs.

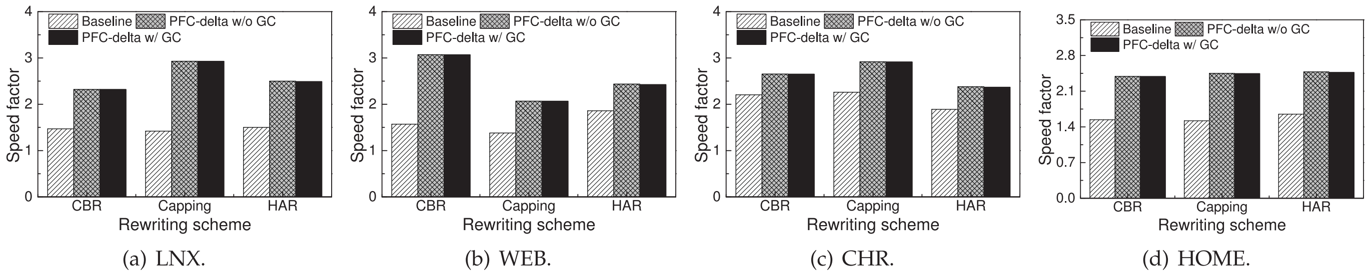


Fig. 11. Speed factors of *Baseline* and *PFC-delta* without and with GC for different rewriting schemes for the four datasets.

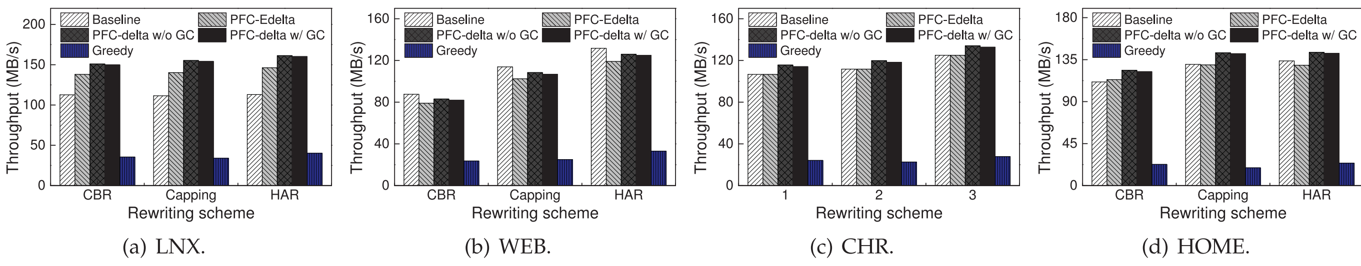


Fig. 12. Backup throughput of *Baseline*, *PFC-Edelta*, *PFC-delta* without and with GC, and *Greedy*, with different rewriting schemes for the four datasets.

VI. Conclusion

Deduplication is often used in backup systems to improve storage space efficiency. Real-world backup products often group files to be protected into larger units (called packed files) before copying them to backup systems. Each file in the packed file will be inserted some metadata before its contents. Since some metadata such as modification time change every time files are packed, introducing substantial unique chunks with unchanged files' contents and changed metadata. Applying delta compression for those similar chunks can further improve storage space efficiency, but at the cost of decreased backup throughput resulting from extra I/Os for fetching base chunks. We observe that the corresponding chunk pairs surrounding identical PFCs are non-identical due to changed metadata but similar to each other. In this article, we propose a high-performance delta compression scheme called PFC-delta to remove redundant data among the aforementioned similar chunks.

For some deduplication-based backup systems, such as DDFS, containers would be accessed for prefetching metadata during deduplication. PFC-delta builds upon this deduplication strategy. It identifies and prefetches chunks stored along with PFCs to serve as base chunks by piggybacking on the I/Os during deduplication. This strategy avoids I/Os for fetching base chunks. On the other hand, to reduce computational overheads for delta encoding, we propose a hash-less delta encoding approach for similar chunk pairs surrounding identical PFCs whose duplicate strings appear exactly in the same positions. The new delta encoding approach is extremely fast because it is devoid of time-consuming fingerprinting and indexing operations. Evaluation results with four real-world datasets demonstrate the effectiveness of our scheme in terms of compression ratio and restore performance and superior write throughput over the typical deduplication-based backup system.

References

[1] G. Wallace et al., "Characteristics of backup workloads in production systems," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST)*. San Jose, CA, USA: USENIX Assoc., Feb. 14–17, 2012, pp. 1–14.

[2] G. Amvrosiadis and M. Bhadkamkar, "Identifying trends in enterprise data protection systems," in *Proc. Conf. USENIX Annu. Tech. Conf.* Santa Clara, CA, USA: USENIX Assoc., Jul. 8–10, 2015, pp. 151–164.

[3] M. Lillibridge et al., "Sparse indexing: Large scale, inline deduplication using sampling and locality," in *Proc. 7th USENIX Conf. File Storage Technol. (FAST)*, vol. 9. San Jose, CA, USA: USENIX Assoc., Feb. 24–27, 2009, pp. 111–123.

[4] F. Guo and P. Efstathopoulos, "Building a high-performance deduplication system," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.* Portland, OR, USA: USENIX Assoc., Jun. 15–17, 2011, pp. 1–14.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on February 29,2024 at 21:56:46 UTC from IEEE Xplore. Restrictions apply.

[5] B. Zhu, K. Li, and R. H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system." in *Proc. 6th USENIX Conf. File Storage Technol. (FAST)*. San Jose, CA, USA: USENIX Assoc., Feb. 26–29, 2008, pp. 269–282.

[6] M. Fu et al., "Design tradeoffs for data deduplication performance in backup workloads," in *Proc. 13th USENIX Conf. File Storage Technol. (FAST)*, vol. 9. Santa Clara, CA, USA: USENIX Assoc., Feb. 16–19, 2015, pp. 331–345.

[7] X. Zou, J. Yuan, P. Shilane, W. Xia, H. Zhang, and X. Wang, "The dilemma between deduplication and locality: Can both be achieved?" in *Proc. 19th USENIX Conf. File Storage Technol. (FAST)*. Santa Clara, CA, USA: USENIX Assoc., Feb. 23–25, 2021, pp. 171–185.

[8] X. Zou, W. Xia, P. Shilane, H. Zhang, and X. Wang, "Building a high-performance fine-grained deduplication framework for backup storage with high deduplication ratio," in *Proc. USENIX Annu. Tech. Conf. (ATC)*. Carlsbad, CA, USA: USENIX Assoc., Jul. 11–13, 2022, pp. 19–36.

[9] Y. Nam, G. Lu, N. Park, W. Xiao, and D. H. Du, "Chunk fragmentation level: An effective indicator for read performance degradation in deduplication storage," in *Proc. IEEE 13th Int. Conf. High Perform. Comput. Commun. (HPCC)*. Banff, Canada: IEEE Comput. Soc. Press, Sep. 2–4, 2011, pp. 581–586.

[10] M. Kaczmarczyk, M. Barczynski, W. Kilian, and C. Dubnicki, "Reducing impact of data fragmentation caused by in-line deduplication," in *Proc. 5th Annu. Int. Syst. Storage Conf. (SYSTOR)*. Haifa, Israe: ACM Assoc., Jun. 4–6, 2012, pp. 1–12.

[11] M. Lillibridge, K. Eshghi, and D. Bhagwat, "Improving restore speed for backup systems that use inline chunk-based deduplication," in *Proc. 11th USENIX Conf. File Storage Technol. (FAST)*. San Jose, CA, USA: USENIX Assoc., Feb. 12–15, 2013, pp. 183–197.

[12] M. Fu et al., "Accelerating restore and garbage collection in deduplication-based backup systems via exploiting historical information," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf*. Philadelphia, PA, USA: USENIX Assoc., Jun. 19–20, 2014, pp. 181–192.

[13] 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 *Proc. 16th USENIX Conf. File Storage Technol. (FAST)*. Oakland, CA, USA: USENIX Assoc., Feb. 12–15, 2018, pp. 309–324.

[14] W. Dong, F. Douglis, K. Li, R. H. Patterson, S. Reddy, and P. Shilane, "Tradeoffs in scalable data routing for deduplication clusters." in *Proc. 9th USENIX Conf. File Storage Technol. (FAST)*. San Jose, CA, USA: USENIX Assoc., Feb. 15–17, 2011, pp. 229–241.

[15] P. Shilane et al., "WAN optimized replication of backup datasets using stream-informed delta compression," in *Proc. 10th USENIX Conf. File Storage Technol. (FAST)*. San Jose, CA, USA: USENIX Assoc., Feb. 14–17, 2012, pp. 1–14.

[16] P. Shilane, G. Wallace, M. Huang, and W. Hsu, "Delta compressed and deduplicated storage using stream-informed locality," in *Proc. 4th USENIX Conf. Hot Topics Storage File Syst*. Boston, MA, USA: USENIX Assoc., Jun. 13–14, 2012, pp. 201–214.

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

[18] D. Trendafilov, N. Memon, and T. Suel, "Zdelta: An efficient delta compression tool," Dept. Comput. Inf. Sci., Polytechnic Univ., Tech. Rep., TR-CIS-2002-02, 2002.

[19] W. Xia, H. Jiang, D. Feng, and Y. Hua, "Similarity and locality based indexing for high performance data deduplication," *IEEE Trans. Comput.*, vol. 64, no. 4, pp. 1162–1176, Apr. 2015.

[20] 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 *Proc. 17th USENIX Conf. File Storage Technol. (FAST)*. Boston, MA, USA: USENIX Assoc., Feb. 25–28, 2019, pp. 129–142.

[21] J. Wu, Y. Hua, P. Zuo, and Y. Sun, "A cost-efficient rewriting scheme to improve restore performance in deduplication systems," in *Proc. 33th IEEE Symp. Mass Storage Syst. Technol. (MSST)*. Santa Clara, CA, USA: IEEE Comput. Soc. Press, May 15–19, 2017, pp. 1–12.

[22] J. Wu, Y. Hua, P. Zuo, and Y. Sun, "Improving restore performance in deduplication systems via a cost-efficient rewriting scheme," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 1, pp. 119–132, Jan. 2019.

[23] Y. Tan et al., "FGDEFRAG: A fine-grained defragmentation approach to improve restore performance," in *Proc. 33th Symp. Mass Storage Syst. Technol. (MSST)*. Santa Clara, CA, USA: IEEE Comput. Soc. Press, May 15–19, 2017.

[24] Y. Tan, B. Wang, J. Wen, Z. Yan, H. Jiang, and W. Srisa-an, "Improving restore performance in deduplication-based backup systems via a fine-grained defragmentation approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 10, pp. 2254–2267, Oct. 2018.

[25] J. Liu, Y. Chai, C. Yan, and X. Wang, "A delayed container organization approach to improve restore speed for deduplication systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 9, pp. 2477–2491, Sep. 2016.

[26] Y. Zhang et al., "Improving restore performance of packed datasets in deduplication systems via reducing persistent fragmented chunks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 7, pp. 1651–1664, Jul. 2020.

[27] X. Lin et al., "Metadata considered harmful to deduplication," in *Proc. 7th UNISEX Workshop Hot Topics Storage File Syst. (HotStorage)*. Santa Clara, CA, USA: USENIX Assoc., Jul. 6–7, 2015.

[28] 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 *Proc. IEEE 36th Int. Conf. Comput. Des. (ICCD)*. Piscataway, NJ, USA: IEEE, 2018, pp. 553–560.

[29] A. Z. Broder, "Identifying and filtering near-duplicate documents," in *Combinatorial Pattern Matching*. San Mateo, CA, USA: Springer, 2000, pp. 1–10.

[30] 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 *Proc. 17th USENIX Conf. File Storage Technol. (FAST)*. Boston, MA, USA: USENIX Assoc., Feb. 25–28, 2019, pp. 121–128.

[31] X. Zou et al., "ODESS: Speeding up resemblance detection for redundancy elimination by fast content-defined sampling," in *Proc. 37th Int. Conf. Data Eng. (ICDE)*. Piscataway, NJ, USA: IEEE, Apr. 19–22, 2021, pp. 480–491.

[32] Y. Zhang et al., "Improving restore performance for in-line backup system combining deduplication and delta compression," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 10, pp. 2302–2314, Oct. 2020.

[33] Y. Zhang, H. Jiang, D. Feng, N. Jiang, T. Qiu, and W. Huang, "Loopdelta: Embedding locality-aware opportunistic delta compression in inline deduplication for highly efficient data reduction," in *Proc. Conf. USENIX Annu. Tech. Conf. (ATC)*. Boston, MA, USA: USENIX Assoc., Jul. 10–12, 2023, pp. 133–148.

[34] W. Xia et al., "Edelta: A word-enlarging based fast delta compression approach," in *Proc. 7th USENIX Conf. Hot Topics Storage File Syst*. Santa Clara, CA, USA: USENIX Assoc., Jul. 6–7, 2015, pp. 7–7.

[35] D. G. Korn and K.-P. Vo, "Engineering a differencing and compression data format." in *Proc. USENIX Annu. Tech. Conf., General Track*, 2002, pp. 219–228.

[36] Y. Cui, Z. Lai, X. Wang, N. Dai, and C. Miao, "QuickSync: Improving synchronization efficiency for mobile cloud storage services," in *Proc. Int. Conf. Mobile Comput. Netw*. Paris, France: ACM Assoc., Sep. 7–11, 2015, pp. 592–603.

[37] Y. Hua, X. Liu, and D. Feng, "Neptune: Efficient remote communication services for cloud backups," in *Proc. IEEE INFOCOM*. Toronto, Canada: IEEE, Apr. 27–May 2, 2014, pp. 844–852.

[38] M. Vrable, S. Savage, and G. M. Voelker, "Cumulus: Filesystem backup to the cloud," in *Proc. 7th USENIX Conf. File Storage Technol. (FAST)*. Santa Clara, CA, USA: USENIX Assoc., Feb. 24–27, 2009, pp. 225–238.

[39] F. Douglis, A. Duggal, P. Shilane, T. Wong, S. Yan, and F. Botelho, "The logic of physical garbage collection in deduplicating storage," in *Proc. 15th USENIX Conf. File Storage Technol. (FAST)*. Santa Clara, CA, USA: USENIX Assoc., Feb. 12–15, 2017, pp. 29–44.

[40] M. Fu. "Destor: An experimental platform for chunk-level data deduplication." GitHub. Accessed: Jun. 10, 2018. [Online]. Available: https://github.com/fomy/destor

[41] W. Xia et al., "FastCDC: A fast and efficient content-defined chunking approach for data deduplication," in *Proc. Conf. USENIX Annu. Tech. Conf. (ATC)*. Denver, CO, USA: USENIX Assoc., Jun. 15–17, 2016, pp. 101–114.

[42] S. Quinlan and S. Dorward, "Venti: A new approach to archival Storage," in *Proc. USENIX Conf. File Storage Technol. (FAST)*. Monterey, CA, USA: USENIX Assoc., Jan. 28–30, 2002, pp. 1–13.

[43] "Linux archives." Kernel. Accessed: Aug. 6, 2020. [Online]. Available: ftp://ftp.kernel.org/

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on February 29,2024 at 21:56:46 UTC from IEEE Xplore. Restrictions apply.

[44] "Sina news." Accessed: Jun. 29, 2016. [Online]. Available: http://news.sina.com.cn/

[45] M. Fu et al., "Reducing fragmentation for in-line deduplication backup storage via exploiting backup history and cache knowledge," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 3, pp. 855–868, Mar. 2016.

Yucheng Zhang received the Ph.D. degree in computer science from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 2017. He is currently an Assistant Professor with the School of Mathematics and Computer Sciences at Nanchang University, Nanchang, China. His research interests include computer storage systems, parallel I/O, and high-performance computing. He has published several papers in major journals and international conferences, including IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, USENIX ATC, FAST, INFOCOM, and IPDPS.

Hong Jiang (Fellow, IEEE) is currently the Chair and the Wendell H. Nedderman Endowed Professor with Computer Science and Engineering Department at the University of Texas at Arlington. Prior to joining UTA, he served as a Program Director with National Science Foundation from January 2013 to August 2015 and he was with the University of Nebraska-Lincoln since 1991, where he was the Willa Cather Professor in computer science and engineering. His present research interests include computer architecture, computer storage systems and parallel I/O, high-performance computing, big data computing, cloud computing, and performance evaluation. He has over 300 publications in major journals and international conferences in these areas, including IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, PROCEEDINGS OF IEEE, ACM-Transactions on Architecture and Code Optimization, ACM-Transactions on Storage, Journal of Parallel and Distributed Computing, ISCA, MICRO, USENIX ATC, FAST, EUROSYS, SOCC, LISA, SIGMETRICS, ICDCS, IPDPS, MIDDLEWARE, OOPLAS, ECOOP, SC, ICS, HPDC, and INFOCOM, and his research has been supported by NSF, DOD, and industry. He is a Member of ACM.

Chunzhi Wang (Member, IEEE) received the Ph.D. degree in computer science from the Wuhan University of Technology. She is currently a Professor with the School of Computer Science at the Hubei University of Technology. Her current interests include big data, peer-to-peer, clouding computing, and network security. She is a member of CCF and ACM.

Wei Huang received the B.E. and M.E. degrees from the Harbin Institute of Technology, Harbin, China, and the Ph.D. degree from Nanyang Technological University, Singapore. He was a Postdoctoral Research Fellow with the University of California, San Diego, CA, USA, and the Agency for Science Technology and Research, Singapore. He is currently a Full Professor with the Department of Computer Science and acts as the Dean of the School of Mathematics and Computer Sciences, Nanchang University, Nanchang, China. He has authored or co-authored more than 100 academic journal or conference papers, including the IEEE TRANSACTIONS ON MEDICAL IMAGING, IEEE TRANSACTIONS ON MULTIMEDIA, MICCAI, and *ACM Multimedia*. His main research interests include machine learning, pattern recognition, computer vision, and multimedia.

Meng Chen received the M.E. degree from Nanchang University, Nanchang, China. He is currently an Assistant Professor with the School of Mathematics and Computer Sciences at Nanchang University, Nanchang, China. His research interests include computer storage systems and parallel I/O and high-performance computing.

Yongxuan Zhang received the Ph.D. degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2020. He is currently an Instructor with Mathematics and Computer School at Yuzhang Normal University. His present research interests include graph processing, high-performance computing, and computer storage systems.

Le Zhang received the M.E. degree from Nanchang University, Nanchang, China. He is currently an Instructor with the School of Mathematics and Computer Sciences at Nanchang University, Nanchang, China. His research interests include computer storage systems and data mining.

Authorized licensed use limited to: University of Texas at Arlington. Downloaded on February 29,2024 at 21:56:46 UTC from IEEE Xplore. Restrictions apply.