# SRSC: Improving Restore Performance for Deduplication-Based Storage Systems

ZUO Chunxue[1], WANG Fang[1], TANG Xiaolan[2], ZHANG Yucheng[1], and FENG Dan[1]
(1. Key Laboratory of Information Storage Systems, Engineering Research Center of Data Storage Systems and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China;
2. 5G Application Product Line, ZTE Corporation, Shenzhen, Guangdong 518057, China)

DOI: 10.12142/ZTECOM.201902009
http://kns.cnki.net/kcms/detail/34.1294.TN.20190507.1052.002.html, published online May 7, 2019
Manuscript received: 2018-06-09

**Abstract**: Modern backup systems exploit data deduplication technology to save storage space whereas suffering from the fragmentation problem caused by deduplication. Fragmentation degrades the restore performance because of restoring the chunks that are scattered all over different containers. To improve the restore performance, the state-of-the-art History Aware Rewriting Algorithm (HAR) is proposed to collect fragmented chunks in the last backup and rewrite them in the next backup. However, due to rewriting fragmented chunks in the next backup, HAR fails to eliminate internal fragmentation caused by self-referenced chunks (that exist more than two times in a backup) in the current backup, thus degrading the restore performance. In this paper, we propose Selectively Rewriting Self-Referenced Chunks (SRSC), a scheme that designs a buffer to simulate a restore cache, identify internal fragmentation in the cache and selectively rewrite them. Our experimental results based on two real-world datasets show that SRSC improves the restore performance by 45% with an acceptable sacrifice of the deduplication ratio.

**Keywords:** data deduplication; fragmentation; restore performance

## 1 Introduction

With a flood of data from companies, networks, emails systems, individual stations and other devices, data deduplication as a type of compression technology is widely employed in the backup systems for saving storage space [1]–[4], [5]. A recent study shows the amount of digital data in the world will exceed 44 Zettabytes in 2020 [6], while EMC reports that more than 90% data are redundant in backup systems [7]. Therefore, data deduplication is an efficient way in space to eliminate redundant data in modern backup systems. Data deduplication splits the file into fixed-size [8] or variable-sized chunks [1], computes the fingerprint of each chunk with hash algorithms (e.g. SHA-1, MD5) and determines whether a chunk is duplicated by looking up a fingerprint index, which records all the fingerprints of non-duplicate chunks in a backup system. If the fingerprint of a chunk has no identical records in the fingerprint index, the chunk is unique and will be written to a read/write storage unit called "container" with fixed size (e.g. 4 MB [9]).

Restore process is of great importance as backup process for the occurrence of disasters such as earthquakes, tsunami and hurricanes [10]. Higher restore performance satisfies the requirements of higher availability of the system. Restoring a backup stream is based on a backup recipe, which includes the metadata information of data chunks, such as chunk fingerprints, chunk ID and chunk size. From the recipe, a container that the chunk locates is read from the disk to a restore cache. When a restore cache is full, it will evict an entire container with a cache replacement algorithm.

Since duplicate chunks are removed from multiple backups, the chunks of a backup stream are physically scattered across different containers, which introduces two types of fragmentation. One is inter-version fragmentation caused by duplicate chunks among multiple versions of the same backup; the other is internal fragmentation caused by duplicate chunks (often called self-referenced chunks [11]) in a single backup. In order to restore a backup stream with fragmentation, multiple containers are read from the disk. Because of the poor random-access performance of hard disk drives (HDDs), fragmentation results in the severely degradation of the restore performance.

To address the fragmentation problem, the state-of-the-art History Aware Rewriting Algorithm (HAR) is proposed to iden-

**This work was supported in part by ZTE Industry-Academia-Research Cooperation Funds, the National Natural Science Foundation of China under Grant Nos. 61502191, 61502190, 61602197, and 61772222) Fundamental Research Funds for the Central Universities under Grant Nos. 2017KFYXJJ065 and 2016YXMS085, the Hubei Provincial Natural Science Foundation of China under Grant Nos. 2016CFB226 and 2016CFB192, and Key Laboratory of Information Storage System Ministry of Education of China.**

tify and collect the duplicate but fragmented chunks in the last backup, and then rewrite them in the next backup. However, for the internal fragmentation, HAR fails to eliminate them because identified internal fragmentation are rewritten in the next backup, rather than being handled immediately. In this paper, we propose an efficient approach called Selectively Rewriting Self-Referenced Chunks (SRSC). The main idea behind SRSC is to maintain a fixed-size buffer, identify whether the self-referenced chunks are fragmented in a buffer and rewrite them in the backup.

The main contributions of this paper include:

(1) As HAR collects fragmented chunks in the last backup and rewrites them in the next backup, we observe that HAR cannot address the internal fragmented chunks caused by self-referenced chunks in a backup.

(2) To reduce internal fragmented chunks in the deduplication-based backup systems, we propose a SRSC scheme to maintain a buffer to simulate a restore cache, identify internal fragmented chunks in the buffer and rewrite them in the current backup.

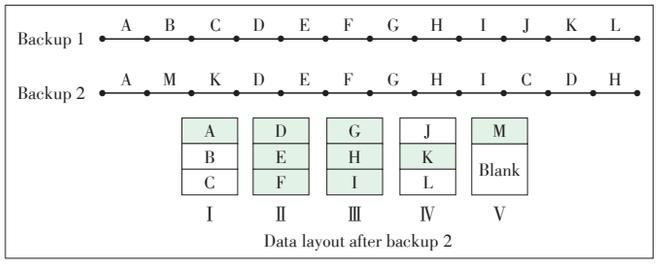
(3) We implement our scheme in the deduplication-based system. Compared with HAR, the experimental results based on two datasets show that SRSC is more efficient in eliminating internal fragmentation and significantly improving the restore performance.

The rest of paper is organized as follows. In Section 2, we review the background for our research. In Section 3, we present our observations to motivate our research. We describes the detailed architecture overview and implementation of our algorithm in Section 4. We present the experimental evaluations with two datasets in Section 5. In Section 6, we present the related work. We conclude our paper in Section 7.

## 2 Background

### 2.1 Fragementation Problem

Fragmentation problem has been received a broad attention in the deduplication-based backup systems. **Fig. 1** shows the fragmentation arises between two backups. Both two backups have 12 chunks. After backup 1, 12 unique chunks of backup 1 are stored in the containers I, II, III, and IV. Since 11 chunks of backup 1 are identical to that chunks of backup 2, only chunk M is stored in the container V after backup 2. The chunks of backup 1 are aggregated in the first 4 containers. However, the chunks of backup 2 are scatter across 5 different containers, which is called fragmentation problem.




▲Figure 1. Fragmentation appears between two backup streams. The shaded chunks represent the chunks referenced by the second backup in each container. The blank areas offer extra space of the half-full containers when a backup completes.

With a 3-container-size Least Recently Used Algorithm (LRU), restoring backup 1 obviously needs to read 3 containers. However, restoring backup 2 needs to read 9 containers. Thus, the restore performance of backup 2 is worse than that of backup 1. This is because reading containers I and II is not efficient and each of the container only includes one useful chunk for backup 2. And then, for restoring a chunk A required by backup 2, we need to read an entire container I. Thus, chunk A is fragmented chunks for backup 2.

### 2.2 History-Aware Rewriting Algorithm

To address the fragmentation problem and improve the restore performance, HAR is proposed. HAR is based on a key observation that two consecutive backups are very similar, and thus history information collection during the last backup is useful for the performance of the next backup.

Specifically, HAR first defines a container's utilization for a backup stream as the fraction of its chunks referenced by the backup. If the utilization of a container is smaller than 50%, the container is regarded as a sparse container that will amplify the read performance. Next, HAR observes that sparse containers of the current backup remain sparse in the next backup. Based on the observation, HAR loads the IDs of sparse containers identified by the last backup, and identifies the sparse containers and rewrites fragmented chunks in the sparse containers. Lastly, HAR collects the IDs of sparse containers as the history information and exploits it for identifying and rewriting fragmented chunks in the next backup. Thus, HAR accurately identifies the fragmented chunks and improves the restore performance.

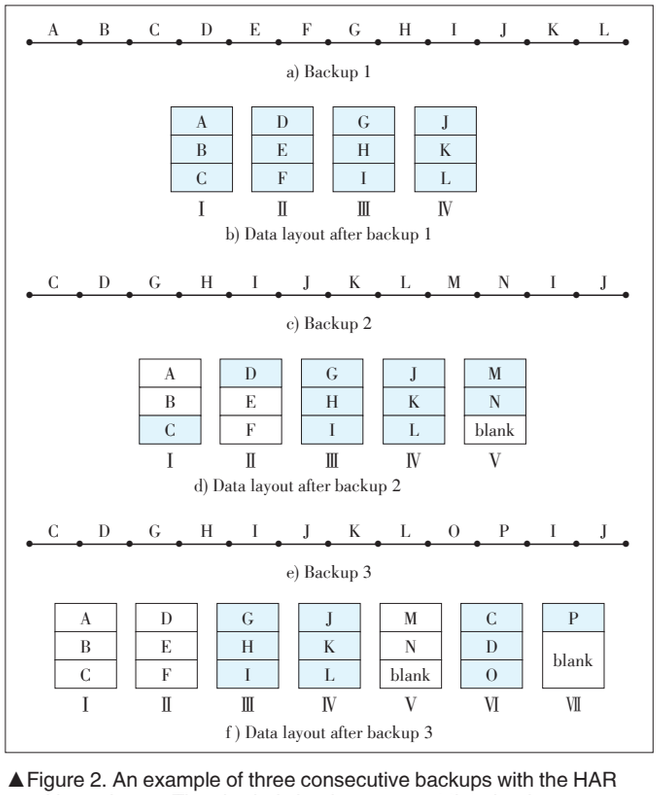
## 3 Motivation

Deduplication is important in backup systems for saving storage space, but introduces fragmented chunks. As described in Section 2.1, the chunks of a backup are scattered in different containers because of the removel of duplicate chunks, which causes the inter-version fragmentation, thus reducing the restore performance. Similarly, for a single backup stream, self-referenced chunks, such as chunk D in backup 2, can also reduce the restore performance. This is because, with a 3-container-size LRU, restoring chunk D two times requires to read the container II two times, which severely reduces the restore performance. Thus, an existing rewriting algorithm such as HAR has been proposed to rewrite inter-version fragmented chunks identified by the last backup to improve the restore performance. However, the design of HAR ignores that internal

fragmentation caused by self-referenced chunks also has a negative impact on the restore performance.

Specifically, at the beginning of a backup, HAR firstly loads the fragmented chunks of the last backup and rewrites them. When the backup finishes, HAR collects information of fragmentation in the sparse containers prepared for the next backup. Obviously, HAR ignores to identify internal fragmentation stemming from self-referenced chunks in a backup, and thus fails to reduce internal fragmentation. In other word, HAR collects fragmented chunks of each backup, but these fragmented chunks come from duplicate chunks between multiple versions of the same backup. Hence, HAR does not consider the impact of internal fragmentation on the restore performance, which hinders HAR from further improving the restore performance.

**Fig. 2** shows an example of the HAR working process, in which all the backup streams have 12 chunks. After backup 1, 12 unique chunks are stored in four containers. Since 10 chunks in backup 2 are duplicate with backup 1, only chunks M and N are written to the containers IV and V. After backup 2, containers' utilization are computed, thus chunks C and D are fragmented in the containers I and II and collected by HAR. When the third backup stream arrives, chunks C and D that have been recorded in backup 2 are rewritten to the container VI in backup 3 (Fig. 2f). We observe that chunks I and J as self-referenced chunks are fragmented because they will lead containers III and IV to be read more than once in a limit




▲Figure 2. An example of three consecutive backups with the HAR rewriting scheme. The shaded chunks represent the chunks referenced by the current backup in each container.

restore cache. However, after HAR algorithm finishes, chunks I and J are still fragmented chunks and not be eliminated. Assume a backup stream has many self-referenced chunks, multiple containers will be accessed during a restore. Thus, HAR increases the number of accessed containers and is an inefficient way to reduce internal fragmented chunks caused by self-referenced chunks.

The above observation motivates us to propose a scheme to reduce internal fragmented chunks. We have proposed a SRSC scheme that maintains a fixed-size buffer to simulate a restore cache, identifies self-referenced chunks whether fragmented in the buffer, and selectively rewrite self-referenced chunks based on computing their containers' utilizations. In such a scheme, internal fragmentation could be accurately identified. Meanwhile, in order to save storage space, we selectively rewrite a part of self-referenced chunks by choosing the low containers' utilization. Therefore, SRSC not only reduces the number of containers, but also improves the restore performance.

## 4 Selectively Rewriting Self-Referenced Chunks

### 4.1 Overview of SRSC

To reduce internal fragmented chunks caused by self-referenced chunks, our SRSC Scheme is proposed to identify self-referenced chunks and selectively rewrite them. As our observation in Section 3, with the increase of backup versions, HAR rewrites fragmented chunks from the last backup but cannot deal with internal fragmented chunks in the current backup. This is because the distance between two self-referenced chunks exceeds the restore cache size during a restore, which causes multiple containers to be read and the restore performance to decline. Thus, a buffer is designed for SRSC to simulate a restore cache. In this buffer, SRSC identifies whether self-referenced chunks are fragmented and then rewrite these fragmented chunks. However, there are at least 20% self-referenced chunks in a backup stream. Rewriting a number of fragmented chunks caused by self-referenced chunks will slow down the backup time and occupy much storage space, which motivates us to rewrite only a part of self-referenced chunks of a backup stream. Thus, SRSC selectively rewrites self-referenced chunks based on the containers' utilization. Except for addressing internal fragmented chunks, we also add HAR rewriting algorithm to the SRSC scheme. This is because HAR takes better advantage of heritability of fragmentation, which is efficient in eliminating inter-version fragmented chunks among multiple versions of backups. Therefore, SRSC can reduce fragmented chunks no matter whether the fragmentation comes from a backup or multiple versions of backups.

### 4.2 Design of SRSC

SRSC is implemented by filter rewriting and selective rewrit-

ing. **Fig. 3** shows the SRSC architecture.

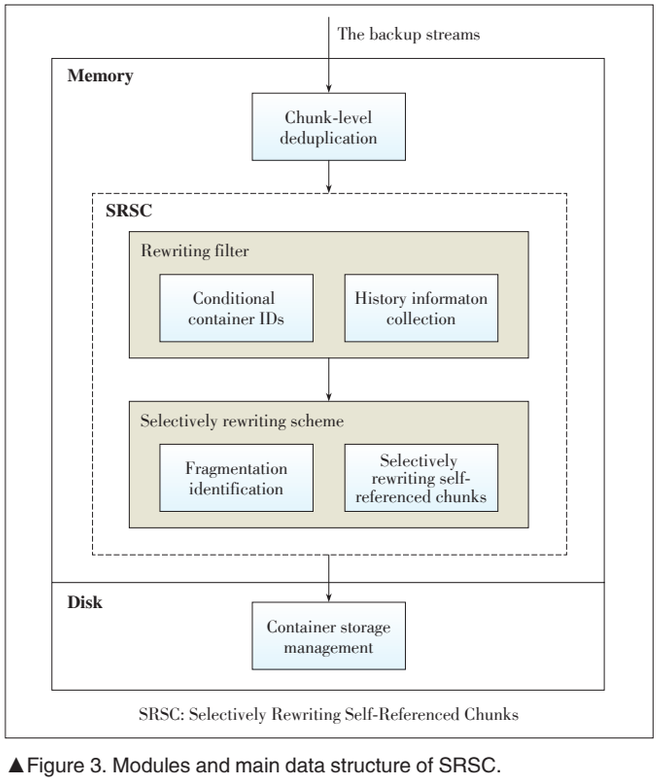
(1) Rewriting Filter

As shown in Fig. 3, the rewriting filter module, stemming from HAR rewriting algorithm, includes two data structures: the conditional container IDs and history information collection. During a backup, the IDs of fragmentation's containers in the last backup are loaded into the conditional container IDs. The rewriting filter module then checks whether the container's IDs of duplicate chunks exist in the conditional container IDs. Before finishing the backup, the history information collection is responsible for collecting the information of fragmented chunks of the current backup, such as containers' IDs of chunks and preparing for the rewriting phase of the next backup.

(2) Selective Rewriting

Whether self-referenced chunks negatively impact containers' replacement in a restore cache depends on the distance between two self-referenced chunks, so selective rewriting is designed to identify internal fragmented chunks and selectively rewrite identified fragmentation. Selective rewriting scheme includes two data structures: fragmentation identification and selectively rewriting self-referenced chunks.

Specifically, fragmentation identification first creates a fixed-size buffer to simulate a restore cache, about 256 MB for experiments. In the buffer, the scheme of selective rewriting identifies self-referenced chunks by checking all the duplicate chunks whether the container ID of the chunks is larger than the total number of the containers stored in the last backup. If so, the chunk is self-referenced, and then fragmentation identification will match the container ID of the chunk with that in the buffer. Finding the identical container's ID means that this self-referenced chunk will not have an impact on the restore performance. Otherwise, the self-referenced chunk is fragmented and needs to be rewritten to the disk. However, considering the storage overheads and deduplication efficiency, the data structure of selectively rewriting self-referenced chunks is used for limiting the number of self-referenced chunks rewritten according to the container's utilization. Thus, the container's ID of this self-referenced chunk is sent to SRSC and is determined whether its chunk needs to be written.




▲Figure 3. Modules and main data structure of SRSC.

We first define a container's utilization threshold as 50%. Next, SRSC computes the utilization of the container that identified fragmentation belongs to. If the container's utilization is smaller than 50%, this self-referenced chunk will be rewritten to the container. Hence, the scheme of selective rewriting not only identifies fragmented chunks by stimulating a restore cache to check if the container ID of the self-referenced chunk can be hold, but also writes a part of the backup stream by using a container's utilization threshold. The work flow of SRSC is shown in **Algorithm 1**.

**Algorithm 1**. SRSC Algorithm

```
1: Initialize a buffer S and define the total number of
   container N;
2: while the backup is not completed do
3:     if the container ID of the chunks is larger than N then
4:         the chunk is a self-referenced chunk
5:         if container ID that the self-referenced chunk
   locates is in S then
6:             Compute the container's utilization C_uti.
7:             if C_uti < 50% then
8:                 the self-referenced chunk will be rewritten to
   the container.
9:             end if
10:        else
11:            Insert the container ID to S.
12:        end if
13:    end if
14: end while
```

## 5 Performance Evaluation

### 5.1 Experiment Setup

(1) Platform

We implement the SRSC algorithm on an opensource deduplication system called Destor [12], which is running on the Ubuntu12.04.2 operating system. The operating system is con-

figured as follows: a 16 GB RAM, a quad core Intel i7-4770 processor at 3.4 GHz. We evaluate SRSC in terms of deduplication ratio and restore performance. Since our paper aims to improve the restore performance of HAR by reducing internal fragmentation, HAR is the baseline of our evaluation, and HAR and SRSC are compared.

(2) Datasets

WEBS, VMDK and FSLHomes datasets are used for evaluations (**Table 1**). Specifically, the WEBS dataset is from the backups of the Sina web with using a tool "wget" to grasp every day. VMDK is a set of virtual machines, including 30 versions. Each backup is 20 GB on average and has 20% self-referenced chunks. The total size of the datasets is about 369 GB. FSLHomes is an open source of traces and snapshots that are collected by file systems and the storage lab and its collaborators, which can be downloaded from the website http://tracer.filesystems.org [13].

(3) Configuration

The backup system divides each dataset into variable-size chunks by using the Rabin chunking algorithm and computes the fingerprints of each chunk with the MD5 hash algorithm. The fingerprint index is stored in memory by default. And the restore cache prefetches and evicts the containers with optimal replacement algorithm [11]. We use the deduplication ratio to evaluate the deduplication efficiency and the speed factor to evaluate the restore performance of SRSC.

## 5.2 Experimental Results and Analysis

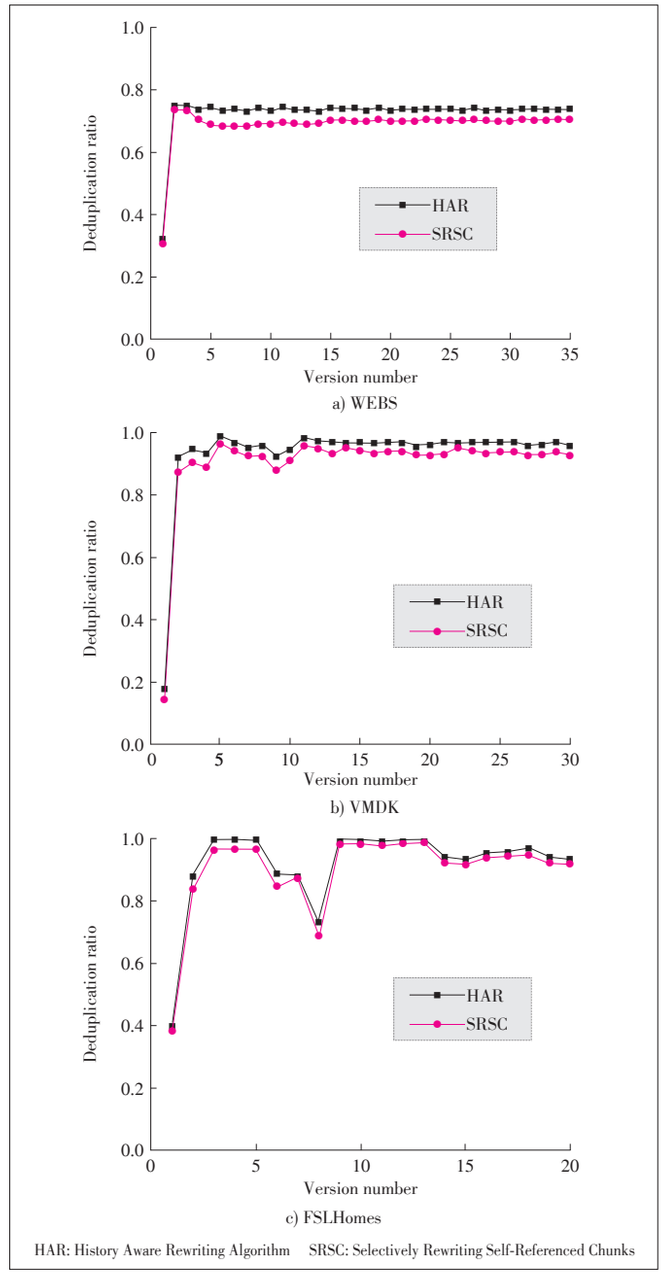
### 5.2.1 Deduplication Ratio

In this evaluation, we use the deduplication ratio as a metric to evaluate the deduplication efficiency. Deduplication ratio is defined as the ratio of the size of duplicate chunks and a file size. **Fig. 4** show the deduplication ratio of HAR and SRSC based on the three datasets. In general, the deduplication ratios of SRSC are lower than HAR. This is because HAR addresses the inter-version fragmented chunks, and thus rewrite fragmented chunks of the last backup. However, compared with HAR, our SRSC scheme rewrites not only inter-version fragmented chunks of the last backup, but also writes a part of self-referenced chunks for addressing the new fragmented chunks caused by selfreferenced chunks in a backup stream. Meanwhile, rewriting fragmented chunks suggests that less duplicate chunks lead to the reduction of the deduplication ratio.

Specifically, for the WEBS dataset, the deduplication ratio of HAR is 73% on average. The deduplication ratio of SRSC is 69% on average, 4% lower than HAR. For the VMDK dataset, the deduplication ratio of HAR is 92% on average. And the deduplication ratio of SRSC is 88% on average, 4% lower than HAR. Similarly, the deduplication ratio of SRSC is 3% lower than HAR for the FSLHomes dataset. Although SRSC's deduplication ratio is a little lower than HAR regardless of the dataset's type, it is reasonable and acceptable.

▼Table 1. Characteristics of three datasets

| Dataset name | Total size (GB) | Version number | Deduplication ratio |
|---|---|---|---|
| WEBS | 105 | 35 | 73% |
| VMDK | 550 | 30 | 92% |
| FSLHomes | 860 | 20 | 91% |



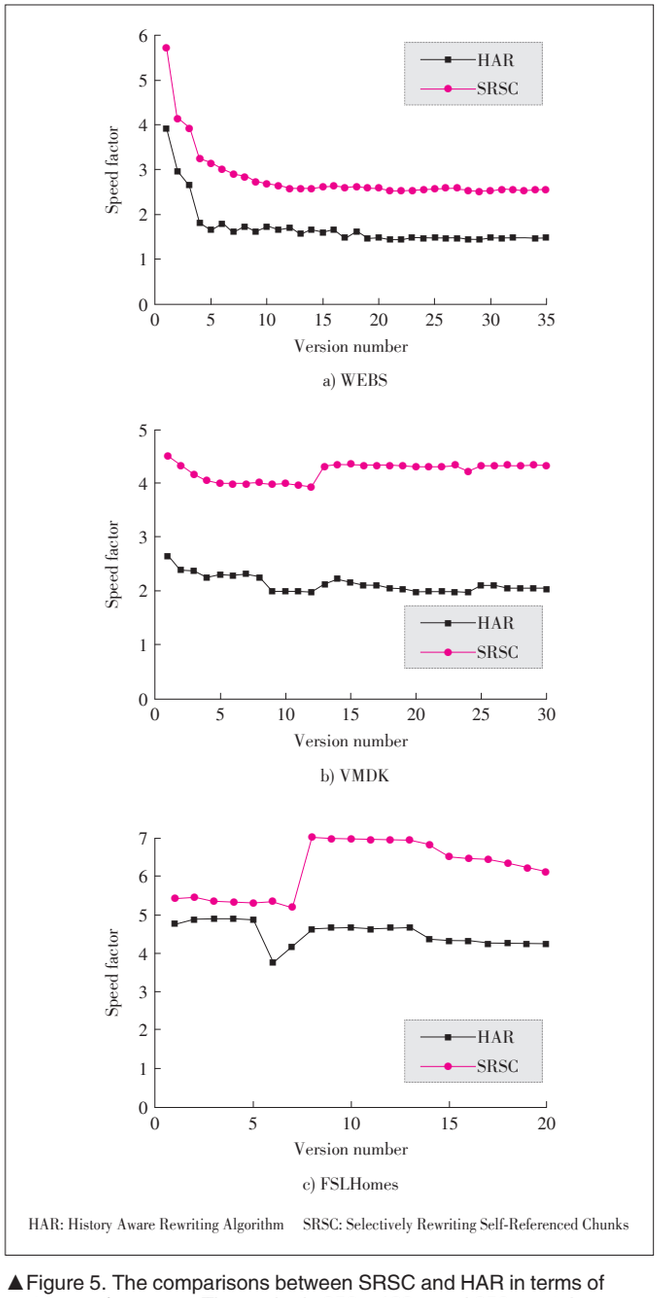

▲Figure 4. The comparisons between SRSC and HAR in terms of deduplication ratio based on the three datasets.

### 5.2.2 Restore Performance

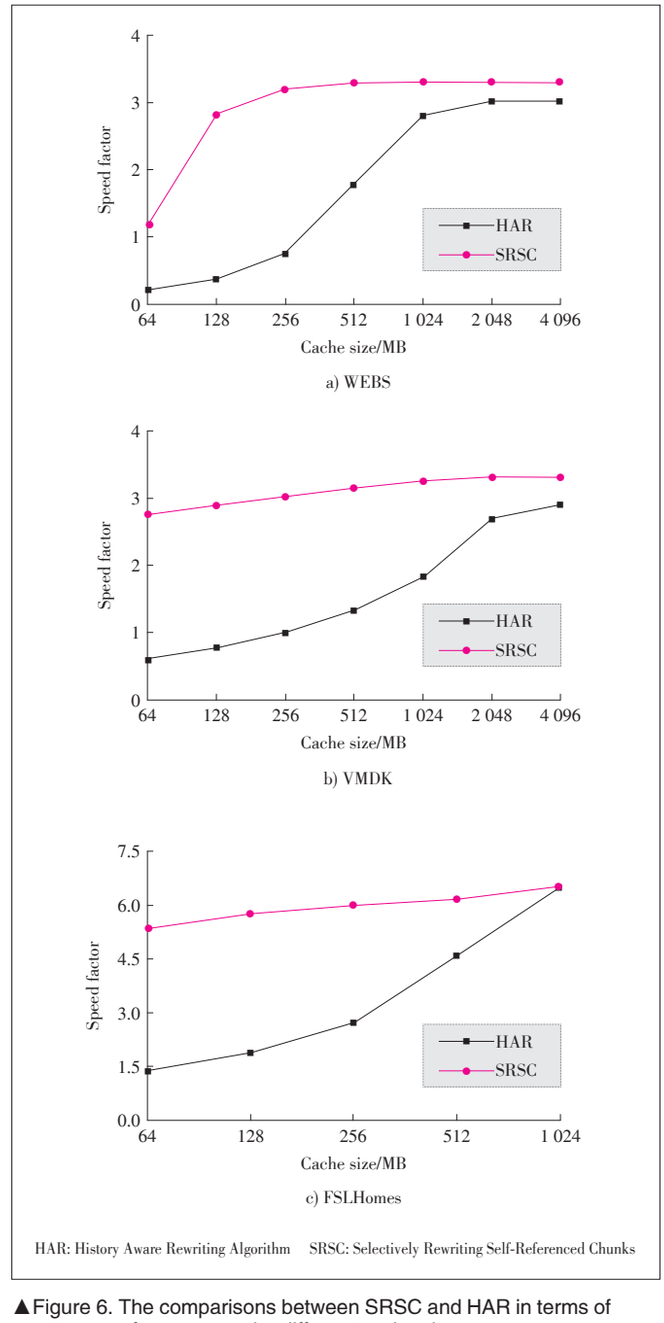
We use the speed factor [11] as a metric to evaluate the re-

store performance. Speed factor represents 1/mean containers read per MB of restored data. **Fig. 5** shows the restore performance between HAR and SRSC based on the three different datasets, which are widely used in the deduplicationbased backup systems [11]. In the figure, SRSC achieves better restore performance than HAR in three data sets. For the WEBS dataset, SRSC outperforms HAR by 40%. For the VMDK dataset, SRSC achieves better restore performance by 48%. For the FSLHomes dataset, SRSC outperforms HAR by 27%. This is because SRSC not only addresses the inter-version fragmentation but also deals with the internal fragmentation. Fragmented chunks are eliminated, which means that the utilization of each container increases. In addition, it also suggests that self-referenced chunks have a negative impact on the restore performance. In general, SRSC achieves higher restore performance than HAR.

**Fig. 6** shows the restore performance between HAR and SRSC under different cache sizes. In WEBS, SRSC significantly improves the restore performance than HAR. Because both inter-version fragmentation and internal fragmentation can de-




▲Figure 5. The comparisons between SRSC and HAR in terms of restore performance. The cache is 256-, 128-, and 256-container-sized in WEBS, VMDK and FSLHomes respectively.




▲Figure 6. The comparisons between SRSC and HAR in terms of restore performance under different cache sizes.

grade the restore performance, SRSC reduces both types of fragmentation.

Note that, with the increase of the cache size, the restore performance becomes higher. This is because a large cache can hold more containers for reducing the number of the containers evicted by the cache replacement algorithm. Moreover, we observe that the restore performance increases more slowly when the speed factor reaches a certain value. This is because the cache size of 1 024 MB can restore a backup stream of WEBS. That is to say, the cache size bigger than 1 024 MB is enough to hold a whole dataset, and thus the speed factor becomes slowly.

### 5.3 Simulated Buffer Size

In this paper, the simulated buffer size is important for the fragmentation identification. This is because a large buffer (e.g., 1 GB size) can hold more containers than a small one at the expense of more memory overheads. However, a small buffer (e.g., 8 MB) identifies more self-referenced chunks as fragmented chunks than a large one. These fragmented chunks will be rewritten, which reduces the deduplication ratio. Thus, SRSC uses a 256 MB buffer, which is reasonable. We find from Fig. 6 that the restore performance keeps relatively stable when the cache size reaches up to 256 MB. Even though the cache size continues to increase, the restore performance improves sightly about 1%−2%. In addition, as shown in Fig. 4, the deduplication ratio does not drop much with using a 256 MB buffer. Based on the observation, a large cache size (more than 256 MB size) is not necessary. Thus, the 256 MB buffer is sufficient for simulating a restore cache.

## 6 Related Work

### 6.1 Data Deduplication

Deduplication is an indispensable component for the backup systems to achieve the goal of saving storage space [15]–[17]. Recent studies for deduplication technology mainly focus on the problem of fast content-defined chunking and large fingerprint indexing. To improve the speed of chunking, some solutions [18]–[21] have been proposed to find an appropriate cut point to achieve a low computation overhead and high chunking throughput for accelerating the process of chunking. To address the large fingerprint indexing, Extreme Binning [22], Silo [23], and Sparse Index [2] exploit the locality of files, the similarity of files, or combines the locality and similarity of the files to improve the performance of fingerprint indexing. Different from above studies, our SRSC algorithm mainly focuses on addressing the fragmentation problem for improving the restore performance.

### 6.2 Restore

The fragmentation problem caused by in-line deduplication has a negative impact on the restore performance. Rewriting algorithms such as context-based rewriting (CBR), capping (CAP), and HAR are proposed to address the fragmentation problem in the deduplication-based backup systems.

CBR [23] uses a sliding window to buffer a small part of the backup stream for identifying fragmented chunks. For each of backup streams, CBR defines the rewrite utility, which is the quotient of the size of chunks that are in the disk not in the backup stream and the total size of the disk, to decide whether it is fragmented. If the rewrite utility of the chunk exceeds the rewrite utility threshold (e.g., 0.5), the chunk is the fragmentation. CAP [24] directly divides the backup stream into fixed-size segments (e.g., 20 MB). In each limited segment, CAP counts the number of referenced containers (N) to recognize fragmented chunks. Assume that N is higher than the threshold value T, the chunks in the containers that hold the least referenced number are regarded as fragmented chunks. HAR [11] is based on the observation that fragmented chunks in the current backup remain fragmented in the subsequent backup. Hence, HAR rewrites the fragmented chunks identified in the last backup, records fragmentation information of the current backup and rewrites them in the next backup.

Rewriting algorithms improve the restore performance with the sacrifice of the deduplication ratio. Some studies take advantage of such characteristic that the read sequence during the restore is identical to the write sequence during the backup to leverage future knowledge for improving the restore performance. The forward assembly area (FAA) [24] maintains an assembly buffer to be filled out the chunks based on the information of a recipe, and thus ensures that the container is read only once in a small space. Limited Forward Knowledge (LFK) uses a backup recipe to improve the restore performance. Park et al. [25] propose a new cache design to evict the container that has the smallest future access by using a lookahead window and maintain a small log to hold evicted containers in order to maximize the cache hit.

In addition to above algorithms, some other schemes are proposed to improve the restore performance. iDeDup [26] exploits spatial locality to selectively eliminate sequential blocks and reduce fragmentation for primary storage systems. RevDedup [27] puts forward a mixed inline and offline deduplication scheme in order to keep the layout of the most up-todate backup as sequential as possible. Mao et al. [28] propose to maintain the chunks with a high reference count on solid state disks (SSDs) of the cloud server to improve the restore performance. CABdedup [29] improves the restore performance by identifying and removing unmodified data transmissions from successive versions of the backup datasets between the client and the cloud.

## 7 Conclusions

In this paper, we propose SRSC, a scheme that simulates a

restore cache to identify whether self-referenced chunks are fragmented and then selectively rewrite them. SRSC not only eliminates internal fragmentation that HAR cannot address, but also further improves the restore performance. Experimental results based on two datasets demonstrate that SRSC improve the restore performance by 45% at an acceptable cost in deduplication ratio.

**References**

[1] ZHU B, LI K, PATTERSON H. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System [C]//6th USENIX Conference on File and Storage Technologies, San Jose, USA, 2008. DOI: 10.1126/science.1164390
[2] LILLIBRIDGE M, ESHGHI K, BHAGWAT D, et al. Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality [C]//7th USENIX Conference on File and Storage Technologies, San Francisco, USA, 2009. DOI: 10.1145/2187836.2187884
[3] DUBNICKI C, GRYZ L, HELDT L, et al. Hydrastor: A Scalable Secondary Storage [C]//7th USENIX Conference on File and Storage Technologies, San Francisco, USA, 2009
[4] FU M, FENG D, HUA Y, et al. Design Tradeoffs for Data Deduplication Performance in Backup Workloads [C]//13th USENIX Conference on File and Storage Technologies, Santa Clara, USA, 2015: 331–344
[5] MEYER D T, BOLOSKY W J. A Study of Practical Deduplication [J]. ACM Transactions on Storage, 2012, 7(4): 1–20. DOI: 10.1145/2078861.2078864
[6] IDC. The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things [EB/OL]. (2014-04). http://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm
[7] WALLACE G, DOUGLIS F, QIAN H, et al. Characteristics of Backup Workloads in Production Systems [C]//10th USENIX Conference on File and Storage Technologies, San Jose, USA, 2012
[8] QUINLAN S, DORWARD S. Venti: A New Approach to Archival Storage [C]//USENIX Symposium on Networked Systems Design and Implementation, Monterey, USA, 2002: 89–102
[9] GUO F, EFSTATHOPOULOS P. Building a High-Performance Deduplication System [C]//USENIX Annual Technical Conference, Portland, USA, 2011
[10] PRESTON W C. Backup and Recovery [M]. Sebastopol, USA: O'Reilly Media, 2006
[11] FU M, FENG D, HUA Y, et al. Accelerating Restore and Garbage Collection in Deduplication-Based Backup Systems via Exploiting Historical Information [C]//USENIX Annual Technical Conference, Philadelphia, USA, 2014
[12] FU M. Destor: An Experimental Platform for Chunk-Level Data Deduplication [EB/OL]. (2014). https://github.com/fomy/destor
[13] TARASOV V, MUDRANKIT A, BUIK W, et al. Generating Realistic Datasets for Deduplication Analysis [C]//USENIX Annual Technical Conference, Boston, USA, 2012
[14] BIGGAR H. Experiencing Data De-Duplication: Improving Efficiency and Reducing Capacity Requirements[R]. The Enterprise Strategy Group, 2007
[15] ASARO T, BIGGAR H. Data De-Duplication and Disk-To-Disk Backup Systems: Technical and Business Considerations [R]. The Enterprise Strategy Group, 2007
[16] XIA W, JIANG H, FENG D, et al. A Comprehensive Study of the Past, Present, and Future of Data Deduplication [J]. Proceedings of the IEEE, 2016, 104(9): 1681–1710. DOI: 10.1109/JPROC.2016.2571298
[17] RABIN M O. Fingerprinting by Random Polynomials [R]. Center for Research in Computing Tech., Aiken Computation Laboratory, Univ., 1981
[18] KRUUS E, UNGUREANU C, DUBNICKI C. Bimodal Content Defined Chunking for Backup Streams [C]//8th USENIX Conference on File and Storage Technologies, San Jose, USA, 2010
[19] AGARWAL B, AKELLA A, ANAND A, et al. Endre: An End-System Redundancy Elimination Service for Enterprises [C]//7th USENIX Symposium on Networked Systems Design and Implementation, San Jose, USA, 2010
[20] ZHANG Y C, JIANG H, FENG D, et al. AE: An Asymmetric Extremum Content Defined Chunking Algorithm for Fast and Bandwidth-Efficient Data Deduplication [C]//IEEE Conference on Computer Communications (INFOCOM), Hong Kong, China, 2015: 1337–1345. DOI: 10.1109/INFOCOM.2015.7218510
[21] BHAGWAT D, ESHGHI K, LONG D D E, et al. Extreme Binning: Scalable, Parallel Deduplication for Chunk-Based File Backup [C]//IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems, London, UK, 2009: 1–9. DOI: 10.1109/MASCOT.2009.5366623
[22] XIA W, JIANG H, FENG, et al. Silo: A Similarity-Locality Based Near-Exact Deduplication Scheme with Low RAM Overhead and High Throughput [C]//USENIX Annual Technical Conference, Portland, USA, 2011
[23] KACZMARCZYK M, BARCZYNSKI M, KILIAN W, et al. Reducing Impact of Data Fragmentation Caused by In-Line Deduplication [C]//ACM SYSTOR, Haifa, Israel, 2012. DOI: 10.1145/2367589.2367600
[24] LILLIBRIDGE M, ESHGHI K, BHAGWAT D, Improving Restore Speed for Backup Systems that Use Inline Chunk-Based Deduplication [C]//11th USENIX Conference on File and Storage Technologies, San Jose, USA, 2013. DOI: 10.1145/2385603.2385607
[25] PARK D, FAN Z Q, NAM Y J, et al. A Lookahead Read Cache: Improving Read Performance for Deduplication Backup Storage [J]. Journal of Computer Science and Technology, 2017, 32(1): 26–40. DOI: 10.1007/s11390-017-1680-8
[26] SRINIVASAN K, BISSON T, GOODSON G, et al. IDedup: Latency-Aware, Inline Data Deduplication for Primary Storage [C]//10th USENIX Conference on File and Storage Technologies, San Jose, USA, 2012: 299–312. DOI: 10.1111/j.1360-0443.2007.01823.x
[27] NG C-H, LEE P P. Revdedup: A Reverse Deduplication Storage System Optimized for Reads to Latest Backups [C]//ACM Asia-Pacific Workshop on Systems, Singapore, Singapore, 2013. DOI: 10.1145/2500727.2500731
[28] MAO B, JIANG H, WU S Z, et al. SAR: SSD Assisted Restore Optimization for Deduplication-Based Storage Systems in the Cloud [C]//IEEE Seventh International Conference on Networking, Architecture, and Storage, Xiamen, China, 2012: 328-337. DOI: 10.1109/NAS.2012.48
[29] TAN Y, JIANG H, FENG D, TIAN L, YAN Z. Cabdedupe: A Causality-Based Deduplication Performance Booster for Cloud Backup Services [C]//IEEE International Parallel & Distributed Processing Symposium, Anchorage, USA, 2011. DOI: 10.1109/IPDPS.2011.76

## Biographies

**ZUO Chunxue** (cxzuo@hust.edu.cn) is currently working toward the Ph.D. degree in computer architecture at Huazhong University of Science and Technology, China. Her research interest is data deduplication and restore performance.

**WANG Fang** received the B.E., M.E., and Ph.D. degrees in computer science and technology from Huazhong University of Science and Technology (HUST), China in 1994, 1997, and 2001, respectively. She is a professor of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, massive storage systems, and parallel file systems. She has more than 40 publications to her credit in journals and international conferences including ACM TACO, SC, MSST, ICPP, ICA3PP, HPDC, and ICDCS.

**TANG Xiaolan** received the M.E. degree in physical electronics from Huazhong University of Science and Technology, China. She is currently a project manager with ZTE Corporation. Her research interests include core network, cloud storage, and 5G technologies.

**ZHANG Yucheng** is currently a Ph.D. student majoring in computer architecture at Huazhong University of Science and Technology, China. His research interests include data deduplication, storage systems, etc. He has several papers in refereed journals and conferences including IEEE-TC, INFOCOM, etc.

**FENG Dan** received the B.E., M.E., and Ph.D. degrees in computer science and technology from Huazhong University of Science and Technology (HUST), China in 1991, 1994, and 1997, respectively. She is a professor and the dean of the School of Computer, HUST. Her research interests include computer architecture, and massive storage systems. She has many publications in major journals and international conferences, including IEEE-TC, IEEETPDS, FAST, USENIX ATC, and MSST.