AN APPROACH TO REDUCE FRAGMENTATION FOR DEDUPLICATION BACKUP STORAGE BY EXPLOITING BACKUP HISTORY AND CACHE KNOWLEDGE

Madhuri Dwarakanath¹, Dr. Ashok kumar P S² Sowmya Naik³

¹Department of Computer Science and Engineering, City Engineering College, Bangalore, India
²Professor & Head, Dept. of CSE, Alems, Bengalore

ABSTRACT:

In backup systems, the chunks of each backup are physically scattered after deduplication, which causes a challenging fragmentation problem. Deduplication results inevitably in data fragmentation, because logically continuous data is scattered across many disk locations. In this work we focus on fragmentation caused by duplicates from previous backups of the same backup set, since such duplicates are very common due to repeated full backups containing a lot of unchanged data. In order to reduce the fragmentation, we propose History-Aware Rewriting algorithm (HAR) and Cache-Aware Filter (CAF). HAR uses historical information in backup systems, and CAF exploits restore cache knowledge. To reduce the metadata overhead of the garbage collection, we further propose a Container-Marker Algorithm (CMA) to identify valid containers instead of valid chunks.

Keywords: Deduplication, fragmentation, HAR, CAF, I-sieve, backup storage, cache knowledge.

[1] INTRODUCTION

Deduplication has become a key component in modern backup systems due to its ability of improving storage efficiency. Deduplication has already been a hot topic in storage for about a decade. The effectiveness of such approach in reducing both time needed to perform backups and storage space required to save them has been widely described and tested. In practice, deduplication has become one of indispensable features of backup systems. Traditionally, performance of a deduplication system is described by the data deduplication ratio, maximal write performance and restores bandwidth. The primary reason for this is data fragmentation caused by deduplication which results in data logically belonging to a recent backup scattered across multiple older backups.

A deduplication based backup system divides a backup stream into variable-sized chunks, and identifies each chunk by its SHA-1 digest, i.e., fingerprint. A fingerprint...
index is used to map fingerprints of stored chunks to their physical addresses. In general, small and variable-sized chunks (e.g., 8 KB on average) are managed at a larger unit called container that is a fixed-sized (e.g., 4 MB) structure. The containers are the basic unit of read and write operations. During a backup, the chunks that need to be written are aggregated into containers to preserve the spatial locality of the backup stream, and a *recipe* is generated to record the fingerprint sequence of the backup. During a restore, the backup stream is reconstructed according to the recipe. The containers serve as the prefetching unit due to the spatial locality.

Since duplicate chunks are eliminated between multiple backups, the chunks of a backup unfortunately become physically scattered in different containers, which is known as fragmentation. The negative impacts of the fragmentation are two-fold:

First, the fragmentation severely decreases restore performance. The infrequent restore is important and a main concern from users. Moreover, data replication, which is important for disaster recovery, requires reconstructions of original backup streams from deduplication systems and thus suffers from a performance problem similar to the restore operation.

Second, the fragmentation results in invalid chunks becoming physically scattered in different containers when users delete expired backups. Existing garbage collection solutions first identify valid chunks and the containers holding only a few valid chunks. Then, a merging operation is required to copy the valid chunks in the identified containers to new containers. Finally, the identified containers are reclaimed. Unfortunately, the metadata space overhead of reference management is proportional to the number of chunks, and the merging operation is the most time consuming phase in garbage collection.

We observe that the fragmentation comes in two categories of containers: sparse containers and out-of-order containers, which have different negative impacts and require dedicated solutions. Both of them hurt the restore performance. Increasing the restore cache size alleviates the negative impacts of out-of-order containers, but it is ineffective for sparse containers because they directly amplify read operations.

[2] RELATED WORK

The fragmentation problem in deduplication systems has received many attentions. iDedup eliminates sequential and duplicate chunks in the context of primary storage systems. Nam et al. propose a quantitative metric to measure the fragmentation level of deduplication systems, and a selective deduplication scheme for backup workloads. SAR stores hot chunks in SSD to accelerate reads. RevDedup employs a hybrid inline and out-of-line deduplication scheme to improve restore performance of latest backups.

The Context-Based Rewriting algorithm (CBR) and the capping algorithm (Capping) are recently proposed rewriting algorithms to address the fragmentation problem. Both of them buffer a small part of the on-going backup stream during a backup, and identify fragmented chunks within the buffer (generally 10-20 MB). The offline approaches traverse all fingerprints (including the fingerprint index and recipes)
when the system is idle. Since recipes need to occupy significantly large storage space, the traversing operation is time consuming.

[3] THE PROBLEM STATEMENT

To illustrate the problem, let us assume full backup of only one file system is saved every week to system with backward pointing deduplication. In such system, the oldest copy of the block is preserved, as is the case with in-line deduplication, because the new copy is not even written.

Usually, a file system is not modified much between two backups and after the second backup many duplicates are detected and not stored again. In the end, the first backup is placed in continuous storage space and all the new blocks of the second backup are usually stored after the end of currently occupied space (see Figure 1).

Such scenario is continued during following backups. After some number of backups, blocks from the latest backup are scattered all over the storage space. This result in large number of disk seeks needed for reading the latest backup and in consequence, a very low read performance (see the restore process of the last backup in Figure 1).

This process can be very harmful to critical restore, because the above scenario is typical to in-line deduplication and leads to the highest fragmentation of the backup written most recently – the one, which will most likely be needed for restore when user data is lost.

[4] DESIGN AND IMPLEMENTATION
HISTORY-AWARE REWRITING ALGORITHM

At the beginning of a backup, HAR loads IDs of all inherited sparse containers to construct the in-memory $S_{inherited}$ structure (inherited IDs in Figure 3). During the backup, HAR rewrites all duplicate chunks whose container IDs exist in $S_{inherited}$. Additionally, HAR maintains an in-memory structure, $S_{emerging}$ (included in collected info in Figure 3), to monitor the utilizations of all the containers referenced by the backup. $S_{emerging}$ is a set of utilization records, and each record consists of a container ID and the current utilization of the container. After the backup concludes, HAR removes the records of higher utilizations than the utilization threshold from $S_{emerging}$. $S_{emerging}$ then contains IDs of all emerging sparse containers.

HAR uses the rewrite limit to determine whether there are too many sparse containers in $S_{emerging}$. (1) HAR calculates an estimated rewrite ratio (defined as the size of rewritten data divided by the backup size) for the next backup. Specifically, HAR first
calculates the estimated size of rewritten chunks for each emerging sparse container via multiplying the utilization by the container size. Then, the estimated rewrite ratio is calculated as the sum of estimated sizes divided by the current backup size, which is approximate to the actual rewrite ratio of the next backup due to the incremental nature of backup workloads. (2) If the estimated rewrite ratio exceeds the predefined rewrite limit, HAR removes the record of the largest utilization in $S_{emerging}$ and jump to step 1. (3) Otherwise, HAR replaces the IDs of the old inherited sparse containers with the IDs of emerging sparse containers in $S_{emerging}$. The $S_{emerging}$ becomes the $S_{inherited}$ of the next backup. The complete workflow of HAR is described in Algorithm 1.

**CACHE AWARE ALGORITHM**

Our observation shows that HAR requires a 2048-container-sized restore cache (8 GB) to outperform Capping in a virtual machine image dataset. The memory footprint is around a half image in size. It is because virtual machine images contain many selfreferences that exacerbate the problem of out-of-order containers. In practice, such a large restore cache could be unaffordable due to concurrent backup/restore procedures.

We develop a Cache-Aware Filter (CAF) to exploit cache knowledge, as an optimization of existing rewriting algorithms. Our key observation is that the sequence of restoring chunks is just the same as the sequence of writing them during the backup. Hence, given an LRU restore cache with a predefined size, we are aware of the runtime state of the restore cache during the backup. CAF simulates the LRU restore cache during the backup using the container IDs of the preceding chunks in the backup stream. For example, when we back up Y, CAF knows the blue container would remain in the restore cache if a 3-container-sized LRU cache were used.
The above figure shows the possible states (SPARSE, OUT OF ORDER, and CACHED) of a duplicate chunk in the hybrid rewriting scheme. The workflow is as follows: (1) The duplicate chunk is checked by HAR. (2) If HAR considers the chunk fragmented, the chunk is marked as SPARSE. Jump to step 9. (3) Otherwise, Capping further checks the chunk (other rewriting algorithms, such as CBR, are also applicable). (4) If Capping considers the chunk NOT fragmented, jump to step 8. (5) Otherwise, the chunk is marked as OUT OF ORDER. CAF checks the chunk’s container ID in the simulated restore cache. (6) If the chunk is expected in the restore cache, it is marked CACHED. Jump to step 8. (7) Otherwise, jump to 9. (8) The chunk is eliminated. Jump to step 1 for the next chunk. (9) The chunk is rewritten. Jump to step 1 for the next chunk.

Based on our observations, only rewriting a small number of additional chunks improves restore performance significantly when the restore cache is small. The hybrid scheme efficiently reduces the cache threshold by a factor of 4 in the virtual machine images. Since the hybrid scheme always rewrites more data than HAR, we suggest to enable the hybrid scheme only in the datasets where self-references are common.

**CONTAINER MARKER ALGORITHM**

We design the Container-Marker Algorithm (CMA) to efficiently determine which containers are invalid. CMA assumes users delete backups in a FIFO scheme, in which oldest backups are deleted first. The FIFO scheme is widely used, such as Dropbox that keeps data of latest 30 days for free users. CMA maintains a container manifest for each dataset. The container manifest records IDs of all containers related to the dataset. Each ID is paired with a backup time, and the backup time indicates the dataset’s most recent backup that refers to the container.

[5] **CONCLUSION**

The fragmentation decreases the efficiencies of restore and garbage collection in deduplication-based backup systems. We observe that the fragmentation comes in two categories: sparse containers and out-of-order containers. Sparse containers determine the maximum restore performance, while out-of-order containers determine the restore performance under limited restore cache. History-Aware Rewriting algorithm (HAR) accurately identifies and rewrites sparse containers via exploiting historical information. We also implement an optimal restore caching scheme (OPT) and propose a hybrid rewriting algorithm as complements of HAR to reduce the negative impacts of out-of-order containers. HAR outperforms the state-of-the-art work in terms of both deduplication ratio and restore performance. The hybrid
scheme is helpful to further improve restore performance in datasets where out-of-order containers are dominant.

To avoid a significant decrease of deduplication ratio in the hybrid scheme, we develop a Cache-Aware Filter (CAF) to exploit cache knowledge. With the help of CAF, the hybrid scheme significantly improves the deduplication ratio without decreasing the restore performance.

Note that CAF can be used as an optimization of existing rewriting algorithms.

The ability of HAR to reduce sparse containers facilitates the garbage collection. It is no longer necessary to offline merge sparse containers, which relies on chunk level reference management to identify valid chunks. We propose a Container Marker Algorithm (CMA) that identifies valid containers instead of valid chunks. Since the metadata overhead of CMA is bounded by the number of containers, it is more cost-effective than existing reference management approaches whose overhead is bounded by the number of chunks.

REFERENCES


Author[s] brief Introduction

Madhuri Dwarakanath
Corresponding Address - No. 33, Sai Krupa, J. P. Nagar 1st Phase, 2nd Cross, C. R. Layout, Near Rajalakshmi Nursing home, Bangalore – 560078. Phone: 9945155145/9538715565