Reverse Deduplication: Optimizing for Fast Restore

Zhike Zhang Preeti Gupta Avani Wildani Ignacio Corderi Darrell D.E. Long
Storage Systems Research Center, University of California, Santa Cruz
{zhikezhang, preetigupta25, avani, icorderi, darrell}@cs.ucsc.edu

1 Introduction

Data deduplication has become an important part of the data storage industry, with most major companies providing products in the space. As additional data is added to a deduplicated storage system, the number of shared data chunks increases. This leads to the fragmentation of the data in the system, which in turn leads to increased seek operations and decreased performance. The challenge for all companies is to provide high performance both at the time of data ingest, and also during data retrieval. In many cases, the primary use of deduplicating storage systems is to provide an alternative to tape-based back-up. For these systems, performance during ingest is important, and the most common retrieval case is the most recent back-up. But due to the nature of existing deduplication algorithms, the most recent back-up is also the most fragmented, resulting in performance issues. We propose to address this issue by changing the way deduplication is done. We developed algorithms to eliminate much of the fragmentation for the most common case, restoring from the most recent back-up.

2 Related Work

As the data in the system increases, so does the fragmentation of the data. The degree to which this occurs is believed to be worse than in a traditional file system. Some have proposed performing periodic defragmentation. Disk defragmentation is more difficult in the presence of deduplication because all references to a chunk must be updated before moving the chunk. Macko, et al. [1] propose the use of back references to solve problems such as finding all file i-nodes in a deduplicated system that reference a block being defragmented. These back references can be used to prevent the file system from having to iterate over the set of all files to find all referenced blocks.

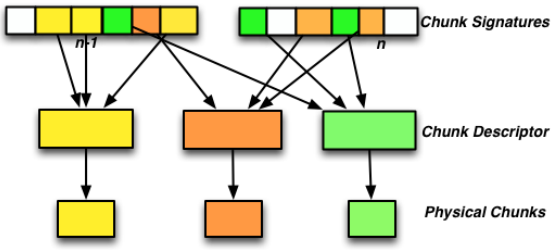

Figure 1: shows chunk signatures pointing to chunk descriptors which are pointing to physical chunks

3 Reverse Deduplication

As additional data is added to a deduplicated storage system, the number of shared data chunks increase. This leads to the randomization of the data in the system, which in turn leads to increased seek operations and decreased performance. This problem occurs regardless of whether the storage system is based on files, segments, or virtual tapes. The existence of shared chunks means that those chunks will with high probability not be contiguous and so require additional seeks. We believe that the best approach is to fundamentally change the way in which deduplication is done in the storage system. Currently, as new data segments are added, they are deduplicated against the existing corpus and duplicate chunks are not stored. The result is that newer segments have greater opportunity to find existing chunks, and so may be increasingly fragmented as the storage system grows. Instead, we propose to invert the deduplication process. Each new segment will be written contiguously, and older data segments that share chunks in the new segment will reference those chunks. The result is that the most common operation, restoring the most recent copy, will be the most efficient.

Efficiently managing the chunks is essential for per-

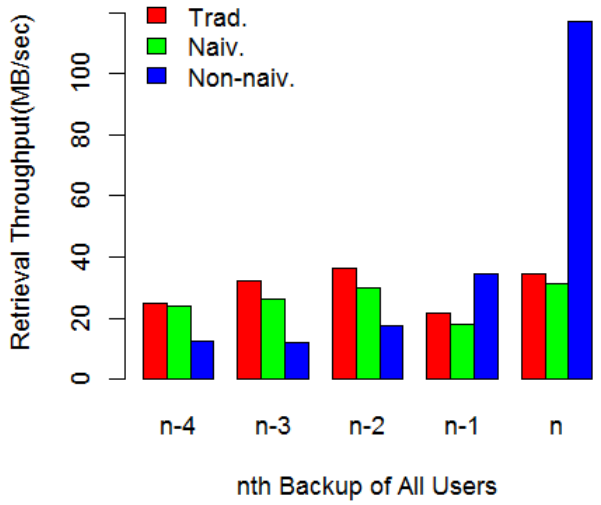

Figure 2: shows the throughput of the restore operation

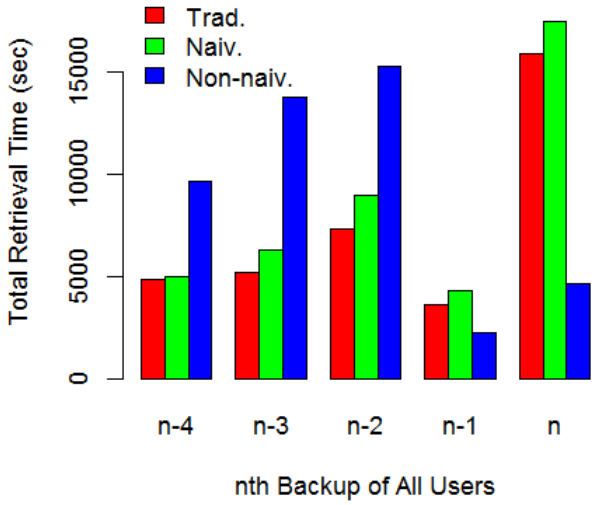

Figure 3: total retrieval time for the most recent backups

formance. Typically, the chunks are reference counted. A chunk in segment $n-2$ might reference a chunk in segment $n-1$ which references a chunk in segment n. A system with so many indirect references will perform poorly when restoring older versions.

One possibility would be to restructure the data as part of the cleaning process. But a better approach would be to introduce chunk descriptors associated with a chunk signature, and reference counted to improve performance. It would have a pointer to the actual physical chunk (shown in Figure 1). In the case when the physical location of the chunk moves, we need to update only the chunk descriptor. Due to the reverse chunking scheme, older data segments will develop holes (portions of the data segment that are no longer referenced). The data accesses will be less contiguous, and so performance for older segments will decrease [2]. Our goal is to keep newer backups more contiguous because they are the most common case for restoration. Older backups will necessarily develop more and more holes.

The solution suggested is an explicit trade off between storage space and retrieving older backups vs. retrieving newer backups.

4 Current Status

We implemented prototypes for Naive Reverse Deduplication, where data is being deduplicated amongst the most recent backups of other users and within itself and Optimized Reverse Deduplication, where data is not being deduplicated amongst the most recent backups of other users and within itself. So for optimized reverse deduplication the most recent backup is not being deduplicated at all till the next most recent backup arrives.

Experiment results are shown in Figures 2 and 3. Figure 2 shows that most recent backup restore throughput for backup systems using optimized reverse deduplication is four times faster than traditional deduplication backup system. Traditional deduplication is sub optimal, as shown in Figure 3, while reverse deduplication degrades linearly going back in time but still performs better than traditional deduplication.

References

[1] P. Macko, M. Seltzer, and K. Smith, "Tracking back references in a write-anywhere file system," in *Proceedings of the 8th USENIX conference on File and storage technologies*, 2010.

[2] R. C. Burns and D. D. E. Long, "Efficient distributed backup with delta compression," in *Proceedings of the fifth workshop on I/O in parallel and distributed systems*, 1997.