Efficient Deduplication in Disk- and RAM-based Data Storage Systems

Andrej Tolič and Andrej Brodnik

University of Ljubljana, Faculty of Computer and Information Science, Slovenia
{andrej.tolic,andrej.brodnik}@fri.uni-lj.si

Abstract. Modern storage systems such as distributed file systems and key-value stores in many cases exhibit data redundancy. The issue is addressed by deduplication, a process of identifying and eliminating duplicate data. While deduplication is typically applied to data stored on disks, the emergence of RAM-based storage systems opens new problems on one hand while being insensitive to some inherent deficiencies of deduplication such as fragmentation. In this paper we present a review of disk- and memory-based deduplication.

1 Introduction

Deduplication (also called redundancy elimination) is a process of identifying and eliminating redundancy in data. Storage systems handle ever increasing volumes of data. However, much of the stored information is duplicate (redundant) data [12]. Often whole files are duplicated in enterprise and cloud environments, while subfile-level redundancy is also common. Not only is storage space preserved. In light of cloud-based services and inter-connectivity, large amounts of data are moved accross networks all over the world. If redundancy could be eliminated, network bandwidth would be preserved [25].

The paper is organized as follows. In section 2, deduplication basics, including types and basic tools, are presented. Section 3 covers techniques for disk-based data deduplication. Techniques are grouped into hash-based, similarity-based and hybrid as described in section 2.1. Additionaly, work specific to virtualization environments is presented in section 3.4. Deduplication techniques for memory-based data are presented in section 4. In section 5 we conclude and present our future work in a way to provide a clear distinction with the reviewed papers.

2 Deduplication Basics

2.1 Deduplication Types

Inline deduplication (also called online) is typically done in the following steps:

1. Data being written to the storage system is analyzed to detect redundancy with respect to data already stored in the system.

2. Redundancy is removed and non-redundant data is written to the system. Data structures holding deduplication metadata are updated accordingly.

In contrast, with **offline deduplication** new data is first written to the system, and then deduplication (redundancy detection and elimination) is done on data "at rest" in batch. Downside to offline deduplication is the need for temporary storage where redundant data is stored before it is deduplicated. Upside is better performance, since deduplication can be performed in background when user access to the data is minimal (eg. during the night).

Redundancy detection and elimination can roughly be divided to **hash-based deduplication** and **similarity-based deduplication**. The emphasis of the first approach is not on using hash functions (despite the name), but on identifying redundant data by *exact-matching* segments using their hash values. The second approach (sometimes called resemblace-based) also uses hash values of data segments, but combines them with other information to form similarity signatures, which results in non-exact *approximate matching*. This is used in one of two ways:

1. Approximately matched similar data is delta encoded.
2. Non-exact similarity matches are faster and therefore used to create multi-tier data structures, where similarity matching is done in the first phase on metadata representing the whole data set and in the second phase used to direct exact-matching on finer granularity. This improves deduplication ratios without increase in computational complexity, which would happen if fine-grained exact matching would be run on large amount of data. An example is Extreme binning approach described in [3].

Hybrid approaches where also proposed and three of them are described later [16, 3, 32]. In hybrid schemes entropy encoding lossless compression is often applied in addition to other techniques.

Basic idea of *delta encoding* (also called differential compression) is to encode one data object as a set of differences (deltas) with regards to other data objects. The main issue with using this technique for deduplication is how to find similar objects so that the compression will be as optimal as possible. This is the reason why this technique is popular in revision control systems and archival tools such as rsync, where reference files are usually identified by same or similar names, or are made explicit by the nature of the application (files as versions of each other).

Another division could be made between **file-level** deduplication and **subfile-level** (sometimes called block-level) deduplication. The first approach is a simple one but effective in certain scenarios [5], especially cloud stores, where many files are stored multiple times. Note, a file is not necessarily a POSIX file but any unit of storage with respect to the storage API. For example, single memory page in memory deduplication (see 4). Authors in [21] collected file system contents from 857 desktop machines at Microsoft over 4 weeks. Testing showed that file-level deduplication achieves about three quarters of the space savings of the most aggressive block-level deduplication for storage of live file systems, and nearly 90% of the savings for backup images.

2.2 Basic Tools

Broder [6] defined notions of resemblance (similarity) and containment between two data objects. The basic idea is to sample objects into sketches and then compute resemblance or containment using sketches.

Rabin fingerprint [28] is a type of a rolling hash function. The name comes from the fact they are used for hashing overlapping regions of data by sliding (rolling) a window over the data. Any hash function could be used for this, but rolling hashes allow faster computation of the new hash value when byte sequences overlap. New hash value can be computed efficiently given only the old hash value, the bits that were removed from and added to the window.

Problem often found in deduplication is how to divide data into parts called *segments* or *chunks* (the two terms are used interchangeably), from which the metadata describing these segments is created. This metadata is stored in a structure which often called *segment index*. One could segment the data into fixed-length segments. However, a simple byte insertion or deletion causes many segments to shift, which is a problem, since in hash-based deduplication segments are checked for exact matching by hashing. To overcome this a solution to the problem of shifting segment boundaries is proposed in [20]. The approach proposes using variable-length segments whose boundaries are selected based on data contents. This approach is also known as *anchoring* (see Figure 1). A rolling

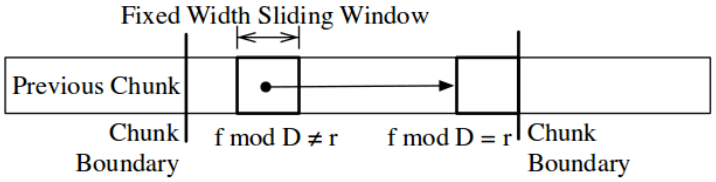

Fig. 1. Anchoring.

checksum (Rabin fingerprint) is computed over a k-byte window which is slided byte by byte through the data. Whenever the lower d bits of the hash equal a specific value r ($f \equiv r \pmod{2^d}$), the k-byte region is marked as a boundary. The value $D = 2^d$ is the divisor and r is the remainder. Assuming uniformly random data, this will happen every 2^d bytes which is the expected average size of a segment. Pathological cases do exist. For example, a long sequence of zero bits would never match a boundary condition. Conversely, k-byte segments all matching the boundary condition could be near each other, thus producing small-size segments. This is usually addressed by setting the minimum and maximum segment sizes as proposed in [25]. If a boundary is found sooner than the minimum size it is skipped. If the boundary is not found when scanning the max-

imum boundary size bytes, a region is selected even if not matching boundary conditions.

To improve the basic anchoring technique a TTTD (two thresholds two divisor) approach was proposed [24]. TTTD finds variable-length segments with smaller length variation than the original anchoring technique. As the name suggests, the algorithm use two thresholds two determine the minimum and maximum sizes of chunks. This is was already proposed in [25]. However, this approach adds a second divisor which is roughly half the size of the main divisor. The second divisor is used to find backup boundaries in case the main divisor fails to do so.

A Bloom filter [4] is a probabilistic data structure used to support set membership queries. They are useful in the context of deduplication, since data structures storing deduplication metadata are often too large to fit into main memory, so disk IO is necessary. Bloom filter-based structures are designed so that they fit into RAM.

3 Disk Data Deduplication

We present several deduplication techniques used of redundancy elimination in disk-based data. This does not mean such techniques cannot be used on memory data.

3.1 Hash-based Techniques

Venti [27] is a block-based write-once storage system, developed as part of the Plan 9 OS from Bell Labs in 2002. Blocks of data are addressed by their fingerprints, which makes Venti a content-addressable storage (CAS). As such, Venti exhibits implicit block-level deduplication, since the same block of data is stored only once. More complex storage services can be build on top of Venti. For example, to store a sequence of blocks (which can represent a file), hashes (addresses) of the blocks in the sequence need to be stored. Venti can be used to store them in the so called "pointer" blocks, which are blocks consisting of a fixed-number of hashes of other blocks. Since only a limited amount of hashes fit into a single block, this process is repeated recursively resulting in a hash (Merkle) tree structure.

Authors in [25] present LBFS, a low-bandwidth file system. LBFS provides the same semantics as well-known network file systems such as NFS or AFS, but uses less network bandwidth to transmit changes between clients and servers. This is achieved by heavy client-side caching and variable-length hash-based deduplication. This is also an early example of using deduplication as one of core WAN optimization techniques. In this case it is specific to the application layer protocol, but WAN optimization devices do the same when deduplicating network packet payloads.

Problem addressed by LBFS is related to the problem addressed by **rsync** [29]. However, rsync only needs to compare two files at a time for similarities which makes the problem easier.

The author [31] presents a technique for optimizing web browsing over slow links. He proposes using delta encoding where the client and the server maintain caches of data objects. When the client wants to retrieve a certain object, the server delta encodes this object using objects that are already in the client's cache as references. A heuristic based on URLs is used to decide on object similarity.

Probably the most cited approach to storage deduplication is described in [34]. Authors present the usage of Bloom filters which has since become standard. The system breaks data into segments using anchoring to produce variable-length content-dependent segments. Segment descriptors are computed that contain SHA-1 fingerprint of the segment, its size and some optional information. A mapping between data objects and segment descriptors is then created. Authors do not go into details, except to explain that some kind of tree is used for the mapping. This is possibly due to commercial use of the scheme. Three acceleration methods are used. First, a *summary vector*, an in-memory structure to reduce disk IO when searching for segments. Summary vector is implemented with a Bloom filter and reduces on-disk segment index accesses when a segment is not duplicated (false positives are possible, but are rare). Second, *locality preserved caching* is implemented. To further reduce on-disk segment index accesses, LPC cache groups together segment descriptors using stream-informed segment layout. So whenever a retrieval from the segment index is made, a group of segment descriptors is transfered to cache. Third, a *stream-informed segment layout* is used, which ensures that spatial locality of segments is preserved. This means the segments are stored on disk in the same order in which they appeared in the workload. This reduces the impact of inherent fragmentation caused by deduplication.

Fragmentation is also addressed in [15]. Authors tweak the inline deduplication system for backup workloads, by "shifting" fragmentation to older backup data.

Researchers at HP Labs proposed **sparse indexing** [18], a technique to cope with on-disk segment index access problem. Instead of using a solution based on Bloom filter similar to the summary vector from [34], they divide the input data stream into sequences of chunks which they call segments, although in other literature, these two notions are the same. They use the TTTD approach [24] for finding segment boundaries, but this is now applied to chunks of bytes instead of single bytes. Each segment is compared for deduplication only with a few of the most similar previous segments. Similarity is identified by deterministicaly sampling a few chunks from within the segment, thus creating a sparse index.

ChunkStash [10] approach stores segment fingerprints on an SSD disk instead of HDD. Cuckoo hashing is used to organize the segment index in RAM. Evaluation of the approach shows that the disk-based scheme performs comparable to the flash-based scheme when there is sufficient locality in the data stream.

Improvements to hash-based deduplication are described in [23]. Authors optimize segment boundary search (anchoring) by improving Rabin's rolling hash algorithm. Additionally, segment index lookup is improved by exploiting access locality.

3.2 Similarity-based Techniques

Several approaches were proposed on how to use delta encoding and similarity detection together, sometimes called DERD (delta encoding with resemblance detection).

According to technique described in [11], large filesets are first divided into clusters. The intent is to group files expected to bear some resemblance. This can be achieved by grouping files according to a variety of criteria including names, sizes, or fingerprints. Authors used name-based clusters to make the processing of a large fileset efficient with regards to memory consumption. Once files are clustered, quadratic complexity techniques can be used to identify good candidate pairs for delta-encoding within each cluster.

Authors in [2] present a similarity-based technique. Since there is no exact matching by hashing, larger segments of data can be used to check for similarities. Similarity signatures are computed to identify possibly similar blocks. A rolling hash is computed with 512-byte sliding window. 4 of the 512-byte subsegments with the largest hash value are selected. Since these hash values are not uniform, they cannot be used as signatures. However, the locations of these 4 512-byte subsegments (they can overlap, but unlikely) are used, shifted by a small amount of bytes, and the hash values of the 4 subsegments starting at this locations are used as signature of the whole segment. The shifting is neccessary to achieve uniform distribution of the 4 hashes. However, no details regarding the index structure are provided, possibly due to commercial use.

3.3 Hybrid Techniques

Paper [16] presents REBL (redundancy elimination at the block level) scheme which combines block-level deduplication (called duplicate block suppression), delta encoding and compression. It is an improvement to the approach from [11] where authors only consider file-level granularity. The algorithm first applies variable-length segment deduplication and then performs resemblance detection on the remaining blocks to identify sufficiently similar blocks on which delta encoding can be used. This makes it possible to efficiently encode segments that differ only slightly, but enough that they are not deduplicated with the block suppression approach. Resemblance detection is optimized by using *superfingerprints*, which are calculated from sets of segment fingerprints, thus reducing fingerprint index size while retaining high resemblance detection. Blocks that aren't handled by any of the above procedure are simply compressed.

Extreme binning [3] technique exploits file similarities in order to deduplicate non-traditional workloads which exhibit low locality. The idea is to store a similarity index of every new file in main memory. This is called a first tier index and contains SHA-1 hash of the whole file as well as ID of the representative chunk (chosen to be the chunk with the minimal hash value) and pointer to the bin stored on disk. Then similar files (the ones having the same representative chunk ID) are grouped into bins stored on disks, which is the second tier index. This eliminates duplicate files in RAM and duplicate chunks inside each bin.

Presidio [32] is a solution for archival storage that similarly to [11] combines multiple deduplication (redundancy elimination) techniques: file-level deduplication, variable-length hash-based deduplication and delta encoding of files based on similarity.

Another hybrid approach is described in [1]. Authors focus on improving segmentation speed with hierarchical fingerprinting based on Merkle (hash) trees. Technique is tweaked to archival workloads.

3.4 VM-specific Deduplication

Variable-length vs. fixed-length deduplication in virtualization environments is inspected in [14]. Somewhat surprisingly, fixed-length chunking achieves better results than variable-length chunking as well as being computationaly less expensive since there is no need for segment boundary detection (anchoring).

In [33], authors introduce Liquid, a deduplication file system designed for large scale VM deployment. The approach uses fixed-sized chunking, with 4KB chunks, which is the same as most OS file systems block sizes. The simplicity of this approach and its efficiency on VM workloads outweigh variable-length chunking.

4 Memory Data Deduplication

Deduplication is not only useful for disk data, but also for memory data. Not much research (compared to disk data) has been done, most of which is focused on virtualization environments, where memory deduplication is also known as **memory page sharing**.

Nevertheless, authors of RAMcloud [26] argue that future storage systems comprised of commodity servers should store data in RAM and not on disks, which shoud only be used for backup. Authors claim that the role of disks must become archival, because it is not possible to access information on disks often enough due to discrepancy between capacity, transfer rate and seek times.

Disco [7] was a project developed to run IRIX OS on a shared-memory multiprocessor computer. Since an obvious problem of excess memory usage occurs when multiple copies of an OS are run, authors developed *transparent page sharing* (memory deduplication). The described approach requires modifications to the guest OS system calls and is therefore application-specific.

Authors in [30] provided a different solution known as *content-based page sharing* developed for the commercial VMware virtual machine monitor. Unlike Disco, this approach does not require any modifications to the guest OS. This approach is in essence an offline (batch) hash-based file-level deduplication where files are memory pages. Several policies are used to control which pages are good candidates and should be scanned, and how often should this be done.

A similar approach is used on Linux, where it is known as Kernel Samepage Merging (KSM) [17]. It was developed as a part of the KVM virtual machine

monitor project. KSM also employs content-based page sharing but uses red-black trees for storing deduplication information instead of hash tables, and a simple 32-bit checksum. This is reportedly [19] due to patent issues with VMware solution [30].

Authors of difference engine [13] apply a similar approach to the one found in [16] to memory pages. They combine hash-based deduplication (often called content-based page sharing in memory context) [30] with delta encoding through resemblance detection [11], thus achieving sub-page-level deduplication.

Memory deduplication in virtual machine environments is not only beneficial since it reduces memory consumption, but it can also save network bandwidth when doing live migration of virtual machines. Technique described in [9] doest just that.

Recently, introspection-based approaches were proposed. XLH (cross-layer hints) approach [22] generates deduplication hints by monitoring guest's access to virtual disk images. Another similar approach is [8] which mostly deals with an issue of efficiently inspecting guest memory.

5 Conclusion and Future Work

We reviewed the most important work in deduplication done in the last decade along with some fundamentals. Most of the techniques in deduplication were developed with backup/archival or virtualization workloads in mind. We notice that work on memory data deduplication is scarce and mostly focused on virtualization.

Our research will focus on the following issues not addressed so far:

- Deduplication (possibly inline) in RAM-based storage systems.
- Subpage-level memory deduplication. Existing systems only deduplicate whole pages.
- Better understanding the effect of different storage APIs on deduplication.

Additionaly, we will investigate the possibility of improving existing techniques, namely:

- Improvements to algorithms for segment boundary scanning and segment indexing in hash-based deduplication.
- Improvements to algorithms for similarity-based deduplication.
- Better handling of inherent fragmentation in disk data deduplication.
- Better approach for distributed data deduplication. We will consider applying deduplication to distributed file systems with beter performance than existing solutions.

References

1. Appaji Nag Yasa, G., Nagesh, P.C.: Space savings and design considerations in variable length deduplication. ACM SIGOPS Operating Systems Review 46(3), 57–64 (Dec 2012)

2. Aronovich, L., Asher, R., Bachmat, E., Bitner, H., Hirsch, M., Klein, S.T.: The design of a similarity based deduplication system. In: Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference on - SYSTOR '09. p. 1. ACM Press, New York, New York, USA (May 2009)
3. Bhagwat, D., Eshghi, K., Long, D., Lillibridge, M.: Extreme Binning: Scalable, parallel deduplication for chunk-based file backup. In: 2009 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems. pp. 1–9. IEEE (Sep 2009)
4. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13(7), 422–426 (Jul 1970)
5. Bolosky, W.J., Corbin, S., Goebel, D., Douceur, J.R.: Single instance storage in Windows® 2000. In: WSS'00 Proceedings of the 4th conference on USENIX Windows Systems Symposium. p. 2. USENIX Association (Aug 2000)
6. Broder, A.: On the resemblance and containment of documents. In: Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171). pp. 21–29. IEEE Comput. Soc (1997)
7. Bugnion, E., Devine, S., Govil, K., Rosenblum, M.: Disco: running commodity operating systems on scalable multiprocessors. ACM Transactions on Computer Systems 15(4), 412–447 (Nov 1997)
8. Chiang, J.H., Li, H.L., Chiueh, T.c.: Introspection-based memory de-duplication and migration. ACM SIGPLAN Notices 48(7), 51 (Aug 2013)
9. Cui, L., Li, J., Li, B., Huai, J., Hu, C., Wo, T., Al-Aqrabi, H., Liu, L.: VMScatter. ACM SIGPLAN Notices 48(7), 63 (Aug 2013)
10. Debnath, B., Sengupta, S., Li, J.: ChunkStash: speeding up inline storage deduplication using flash memory. In: USENIXATC'10 Proceedings of the 2010 USENIX annual technical conference. p. 16. USENIX Association (Jun 2010)
11. Fred Douglis, A.I.: Application-specific Delta-encoding via Resemblance Detection. In: USENIX'03 Proceedings of the 2003 conference on USENIX Annual technical conference (2003)
12. Geer, D.: Reducing the Storage Burden via Data Deduplication. Computer 41(12), 15–17 (Dec 2008)
13. Gupta, D., Lee, S., Vrable, M., Savage, S., Snoeren, A.C., Varghese, G., Voelker, G.M., Vahdat, A.: Difference engine: harnessing memory redundancy in virtual machines. Communications of the ACM 53(10), 85 (Oct 2010)
14. Jin, K., Miller, E.L.: The effectiveness of deduplication on virtual machine disk images. In: Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference on - SYSTOR '09. p. 1. ACM Press, New York, New York, USA (May 2009)
15. Kaczmarczyk, M., Barczynski, M., Kilian, W., Dubnicki, C.: Reducing impact of data fragmentation caused by in-line deduplication. In: Proceedings of the 5th Annual International Systems and Storage Conference on - SYSTOR '12. pp. 1–12. ACM Press, New York, New York, USA (Jun 2012)
16. Kulkarni, P., Douglis, F., LaVoie, J., Tracey, J.M.: Redundancy elimination within large collections of files. In: ATEC '04 Proceedings of the annual conference on USENIX Annual Technical Conference. p. 5. USENIX Association (Jun 2004)
17. KVM: Kernel Samepage Merging, http://www.linux-kvm.org/page/KSM
18. Lillibridge, M., Eshghi, K., Bhagwat, D., Deolalikar, V., Trezise, G., Camble, P.: Sparse indexing: large scale, inline deduplication using sampling and locality. In: FAST '09 Proccedings of the 7th conference on File and storage technologies. pp. 111–123. USENIX Association (Feb 2009)

19. LWN.net: KSM tries again, http://lwn.net/Articles/330589/
20. Manber, U.: Finding similar files in a large file system. In: WTEC'94 Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference. p. 2. USENIX Association (Jan 1994)
21. Meyer, D.T., Bolosky, W.J.: A study of practical deduplication. ACM Transactions on Storage 7(4), 1–20 (Jan 2012)
22. Miller, K., Franz, F., Rittinghaus, M., Hillenbrand, M., Bellosa, F.: XLH: more effective memory deduplication scanners through cross-layer hints. In: USENIX ATC'13 Proceedings of the 2013 USENIX conference on Annual Technical Conference. pp. 279–290. USENIX Association (Jun 2013)
23. Min, J., Yoon, D., Won, Y.: Efficient Deduplication Techniques for Modern Backup Operation. IEEE Transactions on Computers 60(6), 824–840 (Jun 2011)
24. Moh, T.S., Chang, B.: A running time improvement for the two thresholds two divisors algorithm. In: Proceedings of the 48th Annual Southeast Regional Conference on - ACM SE '10. p. 1. ACM Press, New York, New York, USA (Apr 2010)
25. Muthitacharoen, A., Chen, B., Mazières, D.: A low-bandwidth network file system. ACM SIGOPS Operating Systems Review 35(5), 174 (Dec 2001)
26. Ousterhout, J., Parulkar, G., Rosenblum, M., Rumble, S.M., Stratmann, E., Stutsman, R., Agrawal, P., Erickson, D., Kozyrakis, C., Leverich, J., Mazières, D., Mitra, S., Narayanan, A., Ongaro, D.: The case for RAMCloud. Communications of the ACM 54(7), 121 (Jul 2011)
27. Quinlan, S., Dorward, S.: Venti: a new approach to archival storage. In: FAST'02 Proceedings of the 1st USENIX conference on File and storage technologies. p. 7. USENIX Association (Jan 2002)
28. Rabin, M.O.: Fingerprinting by Random Polynomials. Tech. rep., Center for Research in Computing Technology , Harvard University (1981)
29. Tridgell, A.: Efficient Algorithms for Sorting and Synchronization (1999)
30. Waldspurger, C.A.: Memory resource management in VMware ESX server. ACM SIGOPS Operating Systems Review 36(SI), 181 (Dec 2002)
31. Woo, T.: Cache-based compaction: a new technique for optimizing Web transfer. In: IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320). vol. 1, pp. 117–125 vol.1. IEEE (1999)
32. You, L.L., Pollack, K.T., Long, D.D.E., Gopinath, K.: PRESIDIO: A Framework for Efficient Archival Data Storage. ACM Transactions on Storage 7(2), 1–60 (Jul 2011)
33. Zhao, X., Zhang, Y., Wu, Y., Chen, K., Jiang, J., Li, K.: Liquid: A Scalable Deduplication File System for Virtual Machine Images. IEEE Transactions on Parallel and Distributed Systems PP(99), 1–1 (2013)
34. Zhu, B., Li, K., Patterson, H.: Avoiding the disk bottleneck in the data domain deduplication file system. In: FAST'08 Proceedings of the 6th USENIX Conference on File and Storage. p. 18. USENIX Association (Feb 2008)