SADedupe: Skew-Aware Inline Deduplication for Distributed Storage

Binqi Zhang*, Bing Bing Zhou*, Chen Wang†, Dong Yuan*, Albert Y. Zomaya*
*The University of Sydney, Sydney, Australia
†CSIRO, Sydney, Australia
bzha4858@it.usyd.edu.au, {bing.zhou, dong.yuan, albert.zomaya}@sydney.edu.au, chen.wang@csiro.au

Data deduplication has been applied to modern storage systems to improve storage efficiency. Deduplication means to identify and store data only once. Usually, hash functions with low risk of collision such as SHA-1 or MD5 are applied to data blocks to get hash values which represent the contents. Then, the hash values of data blocks are compared. The system assumes two data blocks are identical if their hash values are the same. We call the hash value the "fingerprint" of the data block. Therefore, a fingerprint index is required. When new data arrives, its fingerprint is checked against the index. If there is a match found, the new data is identified as a duplicate and not stored. Initially, systems are equipped with limited memory so the index is stored on disk. This results in the well-known Disk Bottleneck [1] problem as every search results in a disk access. In recent years, systems have started to load the index in main memory. However, the fingerprint index is often too big to fit in memory of a single node.

To achieve higher IO throughput and lower overheads, a two-layer deduplication architecture has been proposed. We design the system on top of the Hadoop Distributed File System (HDFS) [2]. The name node routes the data chunks (MB in size) that are likely to share identical data blocks (KB in size) to the same node by extracting feature ID from minimal hash values according to Broder's Theorem [3]. Then, the deduplication is performed at each node. There are several advantages. First, this adds minimal changes to the distributed file system by only changing the routing algorithm as the chunk size and file recipe meta data can remain unchanged. Second, the deduplication is executed by each node so the file system regards a chunk's writing as complete once it arrives at the node. The process of deduplication can be invisible to the file system. Last but not least, the fingerprint index on each node is much smaller than a global one and can often fit directly into the physical memory. By leveraging the work of CAFTL [4], no fingerprint index is even required on the host of data nodes as the deduplication is performed in the disk.

However, we found one potential problem in this approach: routing chunks with similar contents to the same node may incur load imbalance issues. As time goes on, some nodes become extremely hot. The problem with load imbalance is twofold: First, it may cause significant imbalance in logical storage usage. Second, after deduplication, similar data chunks only have one instance stored on the node. The stored data chunks can have many references. For some popular contents, the reference number can be huge. When the files are read simultaneously, all read requests go to the same referenced chunk, therefore to the same node. The queue on that node can potentially be longer than usual. The overall IO performance can be impeded. Furthermore, this increases the risk of data loss and is a real challenge for fault tolerance. The data loss risk may be easily mitigated by the replication strategy in the file system but the hotness does not go away by simply replicating data. Prior work [5] has paid attention to the first problem which is relatively easier to solve. This work focuses on the second problem, which is often ignored but may lead to performance degradation.

In this work, we carefully analyze the potential impact of data skew in our deduplication system and propose SADedupe as it is aware of the data skew problem and mitigates the impact by trading off a marginal deduplication ratio. It is complementary to the existing file system protocol and can easily be configured. We integrate the reference count of the feature ID (representative hash value) for all chunks into the routing scheme. When the reference count is bigger than a configurable threshold, the upcoming instances with the same feature ID are no longer routed to the original node but to the next one by a modular function. We select the feature ID instead of all hash values of all smaller blocks because the cost of maintaining all hash values becomes impractical for this system. Also, the feature ID is a good representation of the similarity of the entire chunk. Our results show that by capping the reference count and relocating chunks with higher reference counts to more nodes, the storage usage increases by less than 2%. The deviation of both physical and logical storage usage comes down by approximately 15%.

REFERENCES

[1] B. Zhu, K. Li, and R. H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *Fast*, vol. 8, 2008, pp. 1–14.
[2] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–10.
[3] A. Z. Broder, "On the resemblance and containment of documents," in *Compression and Complexity of Sequences 1997. Proceedings*. IEEE, 1997, pp. 21–29.
[4] F. Chen, T. Luo, and X. Zhang, "CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives," in *FAST*, vol. 11, 2011.
[5] W. Dong, F. Douglis, K. Li, R. H. Patterson, S. Reddy, and P. Shilane, "Tradeoffs in scalable data routing for deduplication clusters," in *FAST*, vol. 11, 2011, pp. 15–29.