Data Deduplication on a Distrubuted File System

Keonwoo Oh, Sam Rothstein, Ford Johnstone
Computer Science Department, Swarthmore College, Swarthmore, PA 19081

July 26, 2020

1 Introduction

With the tremendous influx of data, there is an ever increasing need to take measures to reduce the amount of data needed to store, yet maintain data integrity. Data deduplication and data compression are the two main methods for conserving data storage. Data deduplication refers to the elimination of redundant data while data compression refers to the process of encoding information using fewer bits[4]. Given that many files consist of duplicates and that duplicates take up space at the file level, data deduplication could effectively reduce the amount of data needed to store while maintaining data integrity. Data deduplication would be particularly effective in a distributed file system containing large amount of data with many duplicates[4]. However, conventional distributed file systems including Hadoop, Ceph, and Google File System do not have innate support for deduplication[5][2].

Our primary motivation for this project mainly stems from the observation that currently there is no widely adopted deduplication for distributed file system. While there has been attempts to implement deduplication function on a distributed file systems, research in the area is still ongoing. In this context, we propose a distributed file system design that innately supports deduplication. With careful design choices, we hope that the system could perform deduplication without significantly compromising the throughput of the file system. We plan to implement a simple distributed file system based on this model and test them and measure its deduplication ratio as well as read and write throughput, which is quite important for the current applications of distributed file systems. Based on the experimental results, we will be able to evaluate the effectiveness of our model and make suggestions for future research.

2 Related Work

Traditional inline chunk-based deduplication is applicable to distributed file system. Two different methods we examined use this approach. One system proposed by Zhang et al. is based on HDFS, a distributed file system and Hbase, both of which are part of Hadoop, a system that was designed to perform Map Reduce jobs on clusters of commodity hardware. The method is similar to conventional chunk-based method on a local machine in numerous ways: 1) divide data into regular sized chunks 2) compute unique fingerprint of each chunk using crytographic hash function 3) check if the fingerprint is stored in the fingerprint index, supported by HBase. 4) discard the chunk if the finger print is already present, store the chunk in HDFS otherwise[1]. The method also run map reduce jobs for further optimization. Another system, named Droplet, proposed by

Zhang et al. consists of fingerprinting servers and storage servers[7]. Clients of the system send data stream to fingerprinting servers, which divide it into blocks and compute unique block IDs, using cryptographic hash function. Then, the daemon process in fingerprinting servers periodically query storage servers to check if the blocks are already stored and send data stream to storage servers, depending on the result.

One key issue that has is commonly raised against traditional inline, chunk based deduplication in local file system is the disk bottleneck for checking the full chunk index. This slows down the entire data deduplication process. Hence, other probabilistic methods, including bloom filter and sparse indexing, have been proposed as alternatives[3]. However, in case of distributed file system, sparse indexing method do not have memory locality of metadata that is required to speed up the deduplication process. A naive application of bloom filter has scalability issues due to the nature of the data structure. As an alternative, an approach that utilizes Forgetful Updatable Bloom Filter has been proposed for better scalability[6]. The particular method, proposed by Shrayashi et al. combines usage of bloom filter, k mean clustering of syntactic similarity and semantic analysis algorithm to measure data similarity and decide if there are duplicates.

3 Solution

The primary motivation for our work is that the current approaches to deduplication on distributed file system have probabilistic uncertainty. Hence they have limited integration with a distributed file system. The traditional method that solely relies on unique hash function value is highly dependent on the hash function and could unintentionally remove data as a result of hash collision. The probability is extremely small, but is still plausible concern. Due to such unreliability, deduplication is yet to be implemented at the level of distributed file system. Implementation of deduplication at a higher level may compromise performance and increase overhead due to the distributed nature of the file system. For example, the system proposed by Zhang et al. maintain two set of metadata, fingerprint index in HBase and file mapping in HDFS, which is highly costly[1].

The distributed file system we plan to build incorporates deduplication as part of its core functionality. Traditional finger indexing is employed, but for greater system reliability, byte-by-byte comparison is to be made between potential duplicates. The system topology is designed to support locality needed to perform the comparisons without compromising performance. The model we plan to comply by is very similar to HDFS and the original Google File System. The system consists of master and data nodes, where files are split into regular sized blocks and distributed across the data nodes. The master node stores meta data about the file system and the client can fetch the information about the location of a file and directly establish data stream with a particular data node. The system supports write-once-read-many model and provides simple coherency with one-writer-per-file model. Due to time constraints, the model will be greatly simplified and other functions, including fault tolerance, data replication, and authentication will not be supported.

The important feature of our system is that the unique identifier of each data block stored in the master node is based on the hash value computed based on the content of the file. More specifically, the block identifier is a hash value concatenated with another character, by default '0'. The block space is divided into block groups, based on the first letter of the block ID and the blocks in the same block group are stored in the same data node. The client first writes into the buffer. When buffer reaches the size of a block data, the client node computes the hash value of the data and sends it to the master node, which checks the fingerprint index and sends the location of the data node that stores the block group back to the client. The client then establishes data stream with the data node. If there is a block with the same ID, the data node computes byte-by-byte comparison with the existing block of data and the data stream, instead of writing the data. If the data is indeed duplicate, the data node does not store the data and ends the connection. By any chance, if the two blocks have the same ID, but different content, the block is sent again for writing and the latter is assigned block ID that consists of the hash value concatenated with a different character. The block mapping on the master node is updated accordingly.

After building the system, we will throughly test its functionality through a number of experiments to measure deduplication ratio, read, and write throughput. From these results, we could conclude whether our system successfully achieves deduplication without compromising the performance.

4 Experiments

We will run a series of experiments on our system to determine how our implementation of data deduplication impacts the read and write speeds of our distributed file system. Specifically, we will test the system by writing data stream with specific patterns and measure the duplication rate and the read and write throughput. One stream will consist of repeated characters, or repeated segments of data while the other stream will consist of randomly generated characters where we expect no duplication to occur. We will also test different configurations, including, but not limited to, the number of nodes, block size, and buffer size and observe their effects on the read and write output as well as deduplication ratio.

To interpret our results based on the deduplication ratio and read and write throughput. The deduplication ratio measures how well our system implements data deduplication. The deduplication ratio is calculated by dividing the number of bytes inputted into a data deduplication process divided by the number of byte outputted. Read and write throughput would measure how much the deduplication function compromises the file system throughput: whether it is within reasonable bounds.

We may conduct actual benchmark tests with real data if we successfully build a fully functioning file system. However, given our time constraint, this will most likely be accomplished in a future project.

5 Equipment Needed

We will be implementing the system at a user level interface on top of TCP layer. This requires that we have access to local disks of multiple lab machines, rather than the allocated storage on the default NFS. For testing and experiments, we will need access to multiple machines at a certain point in time. We will be using C++, so we do not need any specific software. But as this is expected to be a quite sizable project, general advice and help on specific implementation detail would be greatly appreciated.

6 Schedule

1. Week 1 (04/06-04/12): Finish implementing the client side API. Thoroughly test the functions using “dummy” master node and data node.

2. Week 2 (04/13-04/19): Finish implementing the master node functionality. Thoroughly test the functions using already implemented client side API and dummy data node.
3. Week 3 (04/20-04/26): Finish implementing the data node functionality. Thoroughly test the the fully functioning system.
4. Week 4 (04/27-05/03): Conduct experiments. Write the final report and prepare for presentation.
5. Week 5 (05/03-05/10): Finish writing the report, clean up the source code for submission.

The general objective of this project is to design, implement, and test our original distributed file system that supports data deduplication. With ever increasing amount of data, the topic of the project is increasingly more relevant to our lives. We made unique and novel design choices and we are excited to implement them on our own and see how they turn out. If the experimental results show that deduplication is possible with reasonable throughput, it could indicate that this project may have good prospect for future research. Otherwise, we could analyze the problem again and learn something new.

References

[1] Dongzhan Zhang et al. Data deduplication based on hadoop. In *IEEE CBD*, 2017.

[2] Konstantin Shvachko et al. The hadoop distributed file system. In *IEEE MSST*, 2010.

[3] Mark Lillibridge et al. Sparse Indexing: Large scale, inline deduplication using sampling and locality. In *FAST*, 2009.

[4] Ravneet Kaur et al. Data deduplication techniques for efficient cloud storage management: a systematic review. *The Journal of Supercomputing*, 74:2035–2085, 2018.

[5] Sanjay Ghemawat et al. The google file system. In *ACM symposium on Operating systems principles*, 2003.

[6] Shrayasi Podder et al. A bloom filter-based data deduplication for big data. In *Advances in Data and Information Sciences*, 2018.

[7] Yang Zhang et al. Droplet: A distributed solution of data deduplication. In *IEEE ICGC*, 2012.

Annotated Bibliography

The review provided by Kaur et al. was an extremely helpful resource for obtaining relevant background information and placing the problem in context[4]. It provides a very thorough review of deduplication in the context of cloud computing, which is very relevant to our project. Other than providing information just about large scale deduplication, it provides quite detailed overview of data deduplication in general, providing information about various deduplication techniques, and how they could be categorized accordingly. Our work is very much based on chunk-based inline deduplication.

Deduplication system built by Zhang et al. is one of the works that are very closely related to our work[1]. It is built based on HDFS, distributed file system, and HBase, data base system both of which are part of Hadoop, a general software frame work for a large cluster. The HDFS supports robust fault tolerance and scalability and operates based on master-worker model where the master node maintains metadata about the system while data nodes store the files in regular sized blocks. More specifically, the system Zhang et al. follows traditional chunk-based deduplication with fingerprint index, except at a bigger scale. The files are broken down to chunks, and chunk fingerprint is stored in Hbase. Then, each chunk is stored as a file in HDFS. That the system consists of two different user level layers seemed quite redundant, which motivated our design to integrate them to one.

Droplet is an alternative dedulication system, which is somewhat close to our model[7]. The system is divided into three parts, master node, fingerprinting servers, and storage servers. The master node maintains metadata and the storage servers store the data while finger printing servers are mostly responsible for deduplication process. The client first sends a data block to the fingerprinting servers, which then computes its hash value and tags the data block with it and pushes it to a queue. A daemon periodically collects fingerprints and checks them by querying storage servers. If no deuplicate is found, the data is sent to storage server. If not, the data block is discarded. The system has innate support for data deduplication and it uses hash values as unique block ID, Hence, it is quite similar to our model where file system namespace and deduplication system are well integrated.

Another data deduplocation mechanism integrates various techniques, which include probabilistic method that utilizes bloom filters[6]. Data is hashed to compute unique identifiers, which are then used to detect duplicates. The indices are, however, stored in bloom filters, rather than a database. Bloom filters are probabilistic data structure where false positives are possible for checking whether an element is in the filter. There are shortcomings such as that elements cannot be removed once inserted and that the rate of false positives increases as the filters grow in size. The authors claim to have resolved these issues, using version of the filters. The technique also utilize other techniques of syntax and semantic analysis of data to measure data similarity and detect deduplication.

The paper, published by Yahoo!, discusses the general architecture of HDFS, the file system of Hadoop in great detail[2]. Our model follows this architecture very closely with block-based distributed file system, consisting of name node, which maintains the file directory and name space and worker nodes which store blocks of data. The consistency model is quite simple as HDFS follows write-once-read-many and one concurrent write models. Clients connect to the file system via master node, which responds with information about the location of data block and clients directly establish data stream with the data nodes. Our system is very similar, consisting of a master node, which maintains metadata and worker nodes, which store blocks of data. Major difference, however, is that our system supports deduplication by using hash values as unique block IDs and hence, integrating fingerprinting index and file index to one.