CHAPTER 5

ENHANCING STORAGE EFFICIENCY
BY DEDUPLICATION

5.1 INTRODUCTION

Over the past decade, the amount of data stored in cloud has increased exponentially due to reduction in the disk or storage costs. The data stored needs to be managed and protected. To manage such large data, companies started to replicate the data. Hence it is necessary to increase the efficiency of storage by removing the duplicates and thereby reducing the cost of cloud storage. Deduplication technique is implemented to solve the storage issues. This chapter explains in detail the various issues with respect to deduplication and highlights the essentials of applying the technique to improve the efficiency of cloud store.

The excitement surrounding cloud has evolved into a new level, wherein it has to deal with large datastores in the order of petabytes and zetabytes (Zaffos 2009). Many open source projects support massive data storage in the order of petabytes. The cloud providers use databases that require extreme scalability with improved storage efficiency (Amazon Web Services2013).

(Deng et al 2013) evaluated the performance behavior of memory during compression and deduplication. But the analysis revealed that the
deduplication ratio is compromised to reduce the memory overhead by using fixed chunking than variable chunking.

The evolution of single instance storage saves backup space by identifying files that are repeated. The single instance storage space maintains a full backup of the file that was saved initially. A change to the file or updated copy of the same file is maintained as an incremental backup. The predominant issue identified with such a technique is that it saves all the copies of the same file which have minor differences in it.

A survey suggests that nearly 77 percentage of future backup, use deduplication to improve efficiency and reduce storage costs. Deduplication is the process of eliminating redundant data in a large store using cryptographic techniques. Deduplication ratio achieved during such a process, range from 1:10 to 1:500, resulting in saving disk space and bandwidth to nearly 90% (Russell 2010). The effectiveness of the algorithm used for deduplication depends on the type of data, number of users and retention period.

5.2 RELATED WORK

Many deduplication systems have been created based on identifying duplicates via hash functions in both scientific and industrial fields. However, such systems do not support large size duplicate data as they require an index table of very large size. To improve the performance, the index tables must be cached which gives rise to large cache requirements and substantially increase the cost of the solution (de Carvalho et al 2012; Bobbarjung et al 2006). Similarity Detection was introduced in the context of plagiarism detection (Policriniades & Pratt 2004) and identification of near duplicate documents. It is used in domains such as copy detection, plagarism etc (Peter 2012; Hirscberg 1987). Similarity based deduplication only cresses
information on past data that is likely to be similar and thus more likely to yield good deduplication (Zhang et al 2012a).

Min et al (2011) optimized the deduplication system by adjusting the pertinent factors in fingerprint lookup and chunking. The efficient chunking Incremental Modulo-K(INC-K) algorithm is an optimized Rabin's algorithm which decreases deduplication ratio by 0.66 percent. Zhang et al (2012b) evaluated the data fragments and concluded that the amount of data fragments increases rapidly as the deduplication system grows and for small files, the amount of data fragments increases slowly as the deduplication system grows. Zhou et al (2014) described a deduplication based file communication system with manifest feedback to save network bandwidth by detecting and not sending duplicate "chunks" of the file already in the procession of the receiver. The manifest feedback mechanism accelerates the file communication process over a network via reducing the overheads associated with the interactive duplication detection and query processes.

Min et al (2011) optimized the deduplication system by adjusting the pertinent factors in fingerprint lookup and chunking. The efficient chunking Incremental Modulo-K(INC-K) algorithm is an optimized Rabin's algorithm which decreases deduplication ratio by 0.66 percent. Litwin et al (2012) Variable chunk systems offer better compression, but process data byte-for-byte twice, first to calculate the chunk boundaries and then to calculate the hash. But it suffers from heavy bandwidth of a system. Zhang et al (2012c) evaluated the data fragments and concluded that the amount of data fragments increases rapidly as the deduplication system grows and for small files the amount of data fragments increases slowly as the deduplication system grows.

Zhang et al (2012b) proposed a scheme that used a common data to facilitate fingerprint comparison while reducing the cost and strikes a balance
between local and global deduplication is to increase parallelism and improve reliability through virtualization. Experimental results show the proposed scheme can achieve high deduplication ratio while using a small amount of cloud resources. Similar documents in a large set of unrelated documents have been identified by Fingerprints. Eliminating duplicates aims in identifying blocks by comparing hashes of the object (Litwin et al 2012). Here objects are divided into fixed size blocks and hashed. Fingerprint assists in computing the block size dynamically where the block is divided. Such type is called variable length chunking. It is used to identify duplicates in the backup store (Lu et al 2010). Since Rabin Fingerprint has detailed mathematical foundation and analysis it is widely used in many synchronization and pattern matching algorithms. A comparison of the chunking schemes resulted in the conclusion that CDC outperforms FSC with respect to storage utilization but involves a lot more overheads in maintaining metadata. Delta Encoding has been introduced in versioning system which attempts to encode the difference in two strings. Such a technique stores only the changes made in the consecutive versions and reduces storage overheads. After eliminating identical blocks using variable length chunking, delta technique has been introduced to improve storage utilization. A similar approach is followed to efficiently synchronize replicas. Although data compression techniques eliminate redundancy internal to an object, it reduces the textual data by a factor of 2 to 6 (Rabin1981).

DellDR2400 (Zhu2008) explained the potential to revolutionize the data center. It is proved with different deduplication strategies that target based deduplication with sliding window is the best for cloud needs (Colucci & Benaroch 2010). Deminsfying discussed that deduplication ratio claimed on the server side varies depending upon the retention period during such a process(Colucci2010). Frequency based chunking (Lu et al 2010), an approach for eliminating the metadata overhead was introduced, and it
produces 2.5 – 4 times less number of chunks than variable length chunking. But it requires extra time to find the frequency information of the chunks. Policroniades & Pratt (2004) compare duplication rates for various chunking strategies on a dataset. The findings prove that variable length based chunking performs better than fixed block.

Zhouetal (2014) described a deduplication based file communication system with manifest feedback to save network bandwidth by detecting and not sending duplicate "chunks" of the file already in the procession of the receiver. The manifest feedback mechanism accelerates the file communication process over a network via reducing the overheads associated with the interactive duplication detection and query processes (Zhang et al 2012c; Xia et al 2012) exploits parallelism in achieving deduplication but the ratio achieved in the research work has shown improved results with respect to deduplication time because of the rolling hash algorithm.

5.3 DEDUPLICATION: BACKGROUND

Data Deduplication is evolved to resolve the single instance storage issue. Deduplication allows removing duplicates within files and between files. It saves storage costs due to the fact that it recognizes the differences within files or between files through variable-length blocks. When dealing with storage, the direct cost incurred is by CPU utilization for running the deduplication process and the indirect cost incurred is by the space required for storage, cooling requirements and usage of power.

Data Deduplication is initiated by segmenting the dataset into chunks and computes a Fingerprint or a signature for each. The signatures are stored along with the Index in order to prefetch the chunk whenever required. Chunks stored on the disk are identified by fetching the pointer in the file
allocation table. When accessing the file, the allocation table refers to the pointer from where the blocks can be read. But when the chunk is already available at store, instead of storing it again, a pointer is assigned to the original old chunk. Therefore it is necessary to maintain an index table and a list maintaining chunks. This is referred to as metadata overhead. Repetitive duplicates are removed by inserting multiple pointers to the actual chunk. The actual gain of the overall process involves the difference between the number of duplicates eliminated and the cost incurred in maintaining metadata.

5.4 TECHNIQUES IN DEDUPLICATION

Based on when the deduplication is processed, it is classified as inline or post-process deduplication. When the data is deduplicated before writing it on the disk, it is called inline or pre-processing deduplication. Such a process requires more CPU or memory utilization. In realtime, high demands for processors hinder such a deduplication process. In post-process deduplication, data is processed after writing it on the disk. While checking the file submitted, the storage has to hold the actual data and perform deduplication. This leads to the requirement of greater storage, high end machines and greater bandwidth.

In post-process deduplication, the backup server needs to handle all the process without affecting the server and the client. It leads to network overhead as the duplicate data traverse through the network. This situation is not suitable for an environment in which the network bandwidth is a major constraint. This limitation is resolved by pre-process deduplication where the process runs in a nearby client side storage server before sending the data to the backup server. But, cloud network tends to have large resources which are utilized to perform deduplication during offline mode.
Computation of chunk hashes plays a vital role in the deduplication technique. Various methods exist to break files into individual chunks. Basically chunks created differ in the granularity. They can be classified as fine-grained and coarse-grained chunks. Creation of coarse-grained chunks results in lesser number of chunks that occupy lesser space in the index table, thereby reducing the number of lookups on index while comparing the signature of the chunks. Such an operation consumes very little CPU usage and involves minimal I/O utilization. Conversely, more number of chunks are generated in the case of fine-grained chunking methodology. It basically consumes large index table for maintaining chunks and hence CPU utilization is high. But, fine-grained deduplication process yields good results by saving storage space.

Based on the algorithm used for fine-grained deduplication, it is classified as fixed chunk and variable chunk. In the case of fixed length deduplication, chunks are created by dividing the input data into chunks of equal size and computing the signature. The limitation noticed is that a change in a small chunk will change the rest of the chunks. In the case of fixed-size chunking where the boundaries are fixed by offset rather than content, insertion or deletion of bytes, impacts the resultant chunks. A byte inserted at offset 2, subsequently moves the other bytes by 3 and as a result changing the contents of the entire file. Variable chunk deduplication overcomes this limitation by partitioning the large dataset into variable chunk size. The chunk is defined between two cut points. The cut points are fixed based on the contents of the file. This allows data to be chunked and checked for repetitive blocks. Here, chunks are created based on the block boundary which varies based on the content. Thus change in one part has very little impact on the other chunks of the large dataset. Variable chunk provides a better deduplication ratio than fixed chunk.
5.5 ALGORITHM FOR DEDUPLICATION

Elimination of redundant data involves various algorithms to run at the backend in the real time world. Algorithms basically fall under two categories as deterministic and non-deterministic, based on the kind of output generated for a given input. Basically, to determine whether an algorithm is deterministic or non-deterministic, the output generated is checked for the given hash function or witness function. If the state of the algorithm cannot be determined at every step, such algorithms are called non-deterministic. On the other hand if the state is determined at every step the algorithm is called deterministic algorithm.

Variable chunking algorithm uses rolling hash function which generates hashes very quickly when compared to other algorithms to compute hashes. In variable chunking, the window slides based on the contents of the file. It is necessary to compute hash values for all bytes as the window slides. Instead of calculating the hash values for all bytes as the sliding window moves, the previous value of hashes is used to check the current hash value. As a result bytes when inserted or deleted, doesn’t affect the rest of the chunks.

In this algorithm, the input pattern is interpreted as a integer based on the type of encoding mechanism like ASCII, Unicode etc. The pattern is represented as an array of integers. A hash computation is done in order to convert the array of integers into a single integer. The file or the content of it is verified by performing a simple pattern matching technique. The objective of the process is to find whether the pattern is available in the existing text. The Rabin Karp algorithm is aimed to find the (small file) pattern P in the Text (T), which is a large file. The pattern is represented as a polynomial of degree n-1 over finite field GF(2) as
\[ p(t) = a_1 t^{^{t-1}} + a_2 t^{^{t-2}} + \ldots + a_n \]  \hspace{1cm} (5.1)

Let \( g(t) \) be an irreducible polynomial of degree \( k \), over \( \text{GF}(2) \). The Fingerprint of \( S \) is defined as a polynomial for fixed \( p \) values is given as

\[ F(p) = p(t) \mod g(t) \]  \hspace{1cm} (5.2)

Polynomials used, represent two states as randomly reducible and irreducible. Reducible polynomials are those which are reducible further to polynomials. Whereas irreducible polynomials are those which are not factored to further polynomials. Such irreducible polynomials with higher polynomial degree, achieve higher accuracy. In the next chapter, two polynomial degrees are taken to show the difference and improvement in the results. The simple string matching concept when applied to two files of different size appears in the way represented here. The contents of Pattern \( P=\{975318642\} \) should be checked in the Text = \{3579865321975318642\}. While computing the hash values of the subsets, it is noticed that when the size of the pattern is large, more similarity exists.

Here subsets are formed as

\[ T_0 = \{357986532\} \]
\[ T_1 = \{579865321\} \]
\[ T_2 = \{798653219\} \]
\[ T_3 = \{986532197\} \]
\[ T_4 = \{865321975\} \]
\[ T_5 = \{653219753\} \]
\[ T_n = \{975318642\} \]  \hspace{1cm} (5.3)
A prime is chosen randomly as \( q = 11 \) where \( m < \) the size of pattern. It is noticed from the subsets that, minimally four digits occur in 50% of the subsets which means that as the window slides, a lot of overlap occurs. To save computations, rolling hash has been introduced.

The hash function for a 5 digit number can be represented as

\[
h(t) = (a[0] \times j^{k-1} + a[1] \times j^{k-2} + a[2] \times j^{k-3} + a[3] \times j^{k-4} + a[k-1] \times j^0) \mod m \tag{5.4}
\]

where \( j \) represent the base to represent arithmetic number, \( k \) represent the length of the substring and the array \( a[5] \) represent the contents of the substring whose hash values should be computed. Calculation of hash function in the main text occurs as the window slides. When the window slides, one integer enters on the right hand side (i.e., LSB-Least Significant Bit) and one leaves on the left hand side (i.e., MSB-Most Significant Bit). So the current hash value does not require the computation of hashes at each and every position. Instead, subtracting the least significant bit on one end and adding the most significant bit on other end gives the current hash value gives a generalized equation as

\[
h(t_{i+1}) = a \cdot (h(t_i) - j^{k-1} t[i]) + t[i+k] \mod m \tag{5.5}
\]

where \( h(t_i) \) represents the previous hash computation from which LSB (\( j^{k-1} t[i] \)) is subtracted and MSB (\( t[i+k] \)) is added on the other end. A random irreducible higher polynomial is fixed to calculate the hash values (Fingerprints). During hash computation, collision is the common problem identified. Collision occurs when two dissimilar contents produce same hash value resulting in spurious hit. Such collision is avoided by comparing the string which resulted in spurious hit with the actual string. Such verification process identifies the actual hits among the spurious hits. The running time of the algorithm is improved by using a rolling hash technique.
5.6 EXPERIMENTAL SETUP

The experiments are tested in ubuntu and windowsXP machines. In the experiment, the results are tested for a varied data range from audio, video, image, text data and compressed files. Hashing techniques such as MD5, SHA are tested. Fingerprints are generated and results are tested for varied parameters. Results are generated for variable chunking techniques. Variable chunk size experiments are implemented basically under two types as Rabin Fingerprint for polynomial 54 and Rabin Fingerprint for polynomial 32. To effectively identify the similarity blocks within a file, the results are tested initially by varying the skip bytes during rolling hash technique. It is proved that Rabin Fingerprint using polynomial size 54 remarkably yields good results than Rabin with size 32. In both the cases, the outputs are generated for different values of window, number of bytes skipped and file sizes. A comparison is conducted between the results for selected different strategies such as SHA or MD5, different window size, different skip size in bytes and varied file sizes.

5.6.1 Changing Skip Size

The Rabin-32 Fingerprint is tested for different parameters. The generated results prove that the similarity detection of the contents of the file and the time taken for such deduplication process is relatively less. A series of tests are conducted to fix the average window size. Initial tests are taken with the assumed values of window size. Figure 5.1 shows the performance analysis of different file sizes and the similarity blocks identified using Rabin-32. It is inferred from the Figure 5.1 that as the number of files increases, the similarity blocks increase. For varied size of skip values the similarity blocks has decreased. For each and every chunk the hash values are calculated. The smaller the chunk size, identifying more number of similar chunks is possible. Among all the skip sizes for further processing of results,
the skip size is fixed as 12, during which the similarity chunks are relatively higher than the other values.

**Figure 5.1 Similarity chunks detected by Rabin-32**

In Figure 5.2 the results of Rabin-54 Fingerprint is shown. It indicates that for the same dataset, Rabin-54 gives more number of similarity chunks than Rabin-32.

**Figure 5.2 Similarity chunks detected by Rabin-54**
For varied skip sizes in bytes the deduplication ratio is calculated, which is the ratio of space before deduplication and after deduplication. It is inferred from the result that Rabin-54 performs better and gives 1:910 for skip size 5 and 1:75 for skip size 20. Rabin-32 gives deduplication ratio of 1:8 for skip size 5 and 1:4 for skip size 20.

On the other hand, Rabin-54 gives more number of similarities when compared to Rabin-32 which has crossed nearly 26000 bytes of size similarity for window size 5 for a particular dataset of size 2 MB. But Rabin-32 showed similarity lesser than 3000 bits. From the Figure 5.2, it is noticed that file size of 2 MB has more number of duplicates than the other dataset. Figure 5.3 shown below gives the deduplication ratio of Rabin-32 and Rabin-54. It is evident from the result that space saved due to Rabin-54 is more than Rabin-32. The deduplication ratio is of the order 1:2.

![Graph showing deduplication ratio based on skip size](image)

**Figure 5.3 Deduplication ratio based on skip size**
The time taken to perform the deduplication process is shown in Figure 5.4. The results prove that as file skip byte size increases the numbers of similarities are reduced. Hence the deduplication time which depends on the file content, is relatively less when the similarity blocks are less as shown below.

![Graph showing deduplication time based on skip size](image)

**Figure 5.4 Deduplication time based on skip size**

### 5.6.2 Changing Window Size

The Rabin-32 is tested with different window sizes. The results generated shown in Figure 5.5 reveal that as the window size increases, the number of similarity blocks gets reduced. Hence a smaller window size yields better results than a larger window size. Figure 5.5 reveals the similarity blocks detected when running Rabin-54 Algorithm.
Figure 5.5 Rabin-54 for different window sizes

Figure 5.6 Rabin-32 for different window sizes
The deduplication ratio is calculated for the above two implementations. It is observed from the result that for varied window sizes in bytes, the deduplication ratio is calculated which is the ratio of space before deduplication and after deduplication. It is inferred from the result that the ratio varies from 1:17 to 1:166 for varied sizes for Rabin-32, and 1:39 to 1:923 for Rabin-54 algorithm. The difference in the ratio is due to the similarities noticed which is based on the contents of the file. Figure 5.7 depicts the deduplication ratio of both Rabin-32 and Rabin-54. The deduplication ratio for window size 3 for Rabin-32 and Rabin-54 are 1:166 and 1:923. The window size 8 gave ratio in the order of 1:17 and 1:39 for Rabin-32 and Rabin -54. The time taken for deduplication process is shown in the Figure 5.8. It is observed that the time taken for deduplication is high for window size of 5 which indicates that similar chunks exist of size 5.

![Figure 5.7 Deduplication ratio based on window sizes](image-url)
Figure 5.8 Deduplication time based on window sizes

The Figure 5.8 indicates that after deduplication there is a deviation when window size is 5. It shows from the previous results that time taken for deduplication is relatively high as it has more number of similarity chunks. Hence more number of chunks gives rise to metadata overhead and reduction of redundancy.

5.6.3 Changing Number of Files

The experiment is tested against different file sizes. It is tested for Rabin-32 and Rabin-54. It is distinct that as the file size increases, the reduction in storage space reveals that for the window size of 5, the similarity chunks has reduced. While for Rabin-54, the result has improved where the deduplication ratio is of the order 1:511 and for Rabin-32 the ratio varies from 1:20. Storage space for Deduplication and Rabin-32 Deduplication time while changing the number of files for Rabin-32 and Rabin-54 are shown in the Figure 5.9 and Figure 5.10.
Figure 5.9 Deduplication ratio for varied file sizes

Figure 5.10 Deduplication time for varied file sizes

Figure 5.10 shows deduplication time for Rabin-32 and Rabin-54. It reveals that the deduplication time for polynomial size 32 is less as the process identifies less number of similarities. On the other hand polynomial size 54 gives more number of similarity chunks and as a result, it consumes more time.
5.6.4 Changing File Type

The experiment is tested against different file types. The previous results prove that Rabin -54 showed improved performance. Hence the algorithm is tested against various types of data like audio, video, image, compressed and text data. Figure 5.11 shows similarity chunks for different types of files in bytes.

![Similarity Chunks for Different Dataset](image)

**Figure 5.11  Rabin-54 similarity chunks for file types**

Figure 5.12 shows the deduplication ratio for different datasets. It is evident that the ratio obtained is of the order 1:2 for different sets which proves that it depends on the contents of the file being uploaded. Figure 5.13 gives the deduplication ratio of various datasets.
Figure 5.12 Deduplication ratio for different datasets

Figure 5.13 Deduplication time of various datasets

It is noticed that the time taken for performing deduplication is relatively less for jpg files which shows that compressed file relatively consume less time than uncompressed files.
Figure 5.14  Rabin-54 for different hashing techniques

5.6.5  Changing Hashing Technique

Figure 5.14 showing the output of various hashing techniques. The results are tested for Rabin-54 with two different hashing techniques. From the result, it is proved that MD5 gives more number of similarity blocks than SHA.

5.7  SUMMARY

The algorithm proposed is on pattern matching of the various chunks available, which are changed, based on the random polynomial taken. The comparison of various types of parameters like changing the skip size, window size, and number of files has shown that Rabin-54 gives improved efficiency when compared to Rabin-32.

Moreover the results proved that Rabin -54 with SHA requires less time when compared to MD5. When MD5 is used to check, it yields more number of chunks than SHA and hence the deduplication time is also high. However, the future scope of the research is on improving the storage efficiency by using bloom filters. Lot of research is going on in choosing the
kind of hashing function and the algorithm that solve the problem without collision with high deduplication ratio.

After furnishing the basic requirements of securing large data and improving storage efficiency using Map-Reduce framework, the scalability is the next issue which is discussed in the forthcoming chapter.