EIMDD: Sub-file Level Data Deduplication and Recovery

Tin Thein Thwel
Computer University (Kyainge Tong)
tin.thein.thwel@gmail.com

Abstract

A*s the amounts of storage utilization become the vast, people are being encountered out of storage space in almost every situation. Therefore, they tried to find out the efficient ways to safe storage space. The single instance storage or data deduplication can eliminate multiple copies of the same file and duplicated segments or chunks of data within those files. Hence, data de-duplication becomes an interesting field in storage environments especially in persistent data storage for data centers. Current issue for data deduplication is to avoid full-chunk indexing to identify the incoming data is new, which is time consuming process as it need to match every content of one file to another. This paper, propose an* ***E****fficient* ***I****ndexing* ***M****echanism for* ***D****ata* ***D****eduplication (EIMDD) and recovery system by combining the secure hash algorithm and B+ tree indexing and show experimental results tested on the extents of various file types except media data files. In the proposed system, it will first separate the file into variable-length chunks using Two Thresholds Two Divisors (TTTD algorithm) chunking algorithm. ChunkIDs are then obtained by applying secure hash function to the chunks. The resulted ChunkIDs are used to build as indexing keys in B+ tree index structure. So the searching time for the duplicate chunks of the files reduces from O (n) to O (log n), which can avoid the risk of full chunk indexing. Once the chunks are stored in disk, the system can reconstruct the original file, which is even deleted, using the stored chunks and metadata, whenever the user wants. This meant the recovery ability of the proposed system.*

1. Introduction

Because of the advancement of storage technology, input/output device technology and computer technology, larger fraction of data is now being maintained in digitized form. The amount of data increases every year. The type of data ranges from business data, e.g. medical record, financial record, and legal data to personal one. Most of these data cannot afford to be got lost due to its financial, legal or sentimental value. However, there is also a large amount of duplicated or redundant data in current storage systems. Such duplications may be intended by users for reasons of reliability or availability. Sometimes, it is caused by neglects or mistakes, such identical content or storing the same file into different locations or devices. Still many data duplications are occurred between different files.

To solve these problems, Data de-duplication has get attention in academic and industry. Data de-duplication refers to a kind of approach that uses Lossless Data Compression Algorithms to minimize the duplicate data at the inter file level.

Existing systems solved to reduce storage space with deduplication but most of them didn't consider the risk of full chunk indexing. Albeit some systems can avoid chunk-lookup bottleneck problem, but not fully deduplication. The choice of good searching method can greatly reduce the number of chunk pair evaluations. In this proposed system, we try to use the properties of B+ tree.

B+ tree (BPlusTree) is a type of balanced tree which represents sorted data in a way that allows for efficient insertion, retrieval and removal of records. B+ tree are effectively used in RDBMS, proposed by Bayer and M.C. Creight in 1972, which is originally proposed to replace instead of hashing in RDBMS with B tree [3]. The strength feature of B+ tree is that all paths from the root to leaf are the same length. It is suitable for large amount of data to be indexed so that the system can quickly search duplication data and store instance data if it doesn't find the duplicated data.

As the main goal of proposed research is to reduce the inter-file level duplications, the resulted chunks may take high percentage of random disk accesses, thus we should design an efficient way to save these chunks into actual storage device, which means integrated use of efficient indexing mechanism.

Efficient data de-duplication contains: Dividing each file into variable-length chunks and storing them effectively. We will use the methodology to direct the file chunking using current hash function as cryptographic hash function: SHA1 (Secure Hash Algorithm); Storing the chunks into storage devices efficiently with the help of B+ tree indexing.

The rest of this paper is organized as follows: Section 1 introduces related approaches used to deduplicate the inter-file level redundancies; the

architecture of Efficient Indexing Mechanism for Data Deduplication and the proposed indexing mechanism are presented in Section 3; Section 4 depicts the comparison results of the proposed indexing mechanism; and Section 5 draws the conclusion.

2. Related Work

To the best of our study, B+ tree indexing are only used in relational database system in order to get efficient indexing and not on the inter-file level file storage and indexing with integrated use of hash code (Chunk_ID) as index in data deduplication.

M.Lillibridge et al. described problem statement as chunk-lookup disk bottleneck/full chunks indexing encountered in in-line deduplication and they try to solve using sampling and sparse index and chunk locality[11]. However, they based on assumption that if two segment share one chunk, it is likely to be shared other chunks only limited number of segments are deduplicated and can't do fully deduplication as sometimes can store duplicate chunks.

Sridhar Ramaswamy pointed out the indexing problems in constraint and temporal data [13]. They proposed to uses B+ tree but their work had done on database, solution is optimal for query, but doesn't have good worst-case bounds for updating. Walter Santos solved the problem of identification of replicas in database with parallel deduplication algorithm using filter-stream model [14] and only considered for databases and not for sub-file level of the file which have their respective hash code.

Daniel P.Lopresti described the issues related to the detection of duplicates in document image databases and proposed four algorithm using edit distance [5]. But they can solve only for image database specific.

Jared, D. et al. find out that accurate rule-based deduplication requires significant manual tuning of both rules and thresholds. They proposed learning – based information fusion using SVM but done on database record, specific to set of deduplication rules [9]. If other set of rules use, match accuracy values will decrease.

Z. Benjamin et al. proposed Summary Vector (Bloom Filtering) and Locality Preserved Caching to avoiding the disk bottleneck in the Data Domain Deduplication File System [18]. Their assumption is that if the last time encountered chunk A, it was surrounded by chunks B, C, D, then next time encounter A, it is likely to encounter B, C or D nearby. So, their system can avoid full chunk indexing. Because of this assumption, this system is not fully deduplication and sometimes can store duplicates.

3. System Architecture

This section explains the overall architecture of the proposed system.

3.1. Proposed Framework

The proposed framework for the data deduplication is as shown in Figure 1. The system will first divide the file into chunks using TTTD chunking algorithm. After that, the variable-length chunks are input into Chunk_ID generator to obtain obtained the ChunkIDs. The resulted ChunkIDs are used to build as indexing keys in B+ tree index structure.

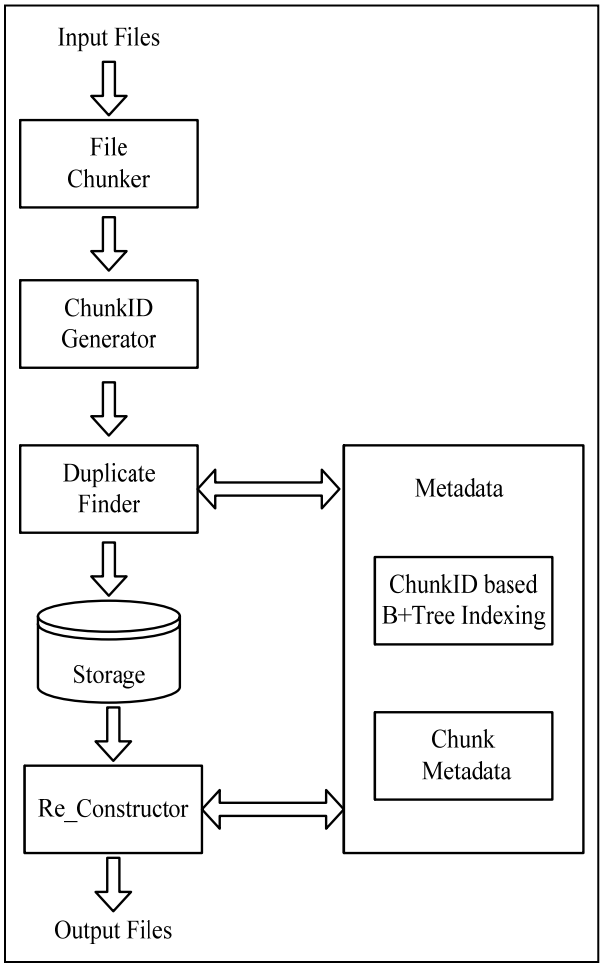

Figure 1. Proposed system architecture

3.1.1. System Components Description

The system includes the major components: File Chunker, Chunk_ID Generator, Duplicate Finder, Metadata, Storage, and also include the Re-Constructor. Input file format may include .PDF, .html, .txt, .doc, etc. Input files are separated into variable

length chunks by File Chunker. The Chunk_ID Generator uses the resulted chunks from the File Chunker to generate Chunk_ID by applying SHA1, which is secure hash function. By using the Chunk_ID, Duplicate Finder checks whether that Chunk_ID is already exist or not in the Metadata. Proposed B+ tree indexing mechanism is built in the Metadata. The content of the chunk data are stored in the storage space.

File Chunker

File Chunker uses the Two Thresholds Two Divisors (TTTD Algorithm) Algorithm to segment the input files as the chunks.

Chunk_ID Generator

To generate the Chunk_ID, the Chunk_ID generator uses SHA1 which produces 160 bits signature for each chunk.

Duplicate Finder

The Duplicate Finder finds the duplicate Chunk_ID in the existing B+ tree indexing structure with the incoming Chunk_ID. The detail process of the Duplicate Finder is describes in next Section 4.1.4.

Metadata

The metadata maintain the B+ tree structure of the already stored files information including Chunk_ID, file information. The proposed indexing mechanism using B+ tree properties is depicted in Section 4.3.

3.1.2. File Chunker

For data deduplication, chunking technique is commonly used. Figure 2 shows the general process of file chunker.

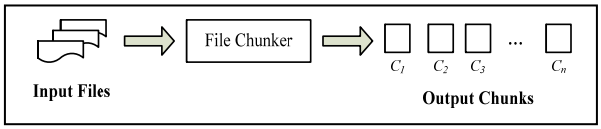

Figure 2. General process of File Chunker.

In order to segment the input files as the chunk, the TTTD algorithm is used. Whatever metadata scheme is used, the costs of metadata and process of hashing depend on the number of chunks generated for a file. The smaller the chunk size, the better the duplication. However, large number of small chunks led to overhead of chunk hash, index the Chunk_ID and metadata increases. On the other hand, large chunks can reduce one's chances of identifying duplicate data.

In the proposed system, it use Two Threshold Two Divisor algorithm (Eshghi, K. 2005) [7], which uses applies a minimum and maximum size threshold when setting the boundaries of every chunk. If the boundary condition using one of the divisors evaluates to be true, this boundary is held as a backup. This is done in case the next boundary condition, using the second divisor, does not evaluate to be true for a long time resulting in a large chunk. Therefore, it can avoid small chunking and large chunking.

3.1.3. Chunk_ID Generator

To generate the Chunk_ID, hashing algorithm can use in order to identify the identical chunks. In this system, the Chunk_ID Generator uses SHA1 which produces 160 bits signature for each chunk and also a collision-resistant function. It can reduce the risk of finding identical information in the file.

3.1.4. Duplicate Finder

Indexing plays an important role in deduplication process. In this work, we try to construct and search the B+ tree like structure for indexing.

The algorithm for the duplicate finder is as follow:

DuplicateFinder(chunkID,chunkIDMetadata,Chunk)

Begin

found ← searchInBPlusTree(chunkID)

if found in BPlusTree then
get Address of that chunkID's content
updateFileMetaData(chunkIdAddress, chunkIDMetadata)

else
updateBPlusTree(chunkID)
updateFileMetaData(chunkIdAddress, chunkIDMetadata)
store(chunk)
endif

End

3.2. Proposed B+ Tree Indexing Mechanism

The resulted Chunk_ID generated from the Chunk_ID Generator are used to construct as B+ tree index structure and maintains as metadata. By using the advantage of B+ tree properties, the optimal search time O(log n) which is more efficient than the full chunk indexing O(n). The proposed indexing mechanism is as shown in Figure 2.

Where,

$Chunk\_ID_n$	– Hash Code
Chunk_ID's Metadata	–file name, file type, chunk number, offset of the chunk in that file, chunk length
$Chunk_n$	–Content of chunk

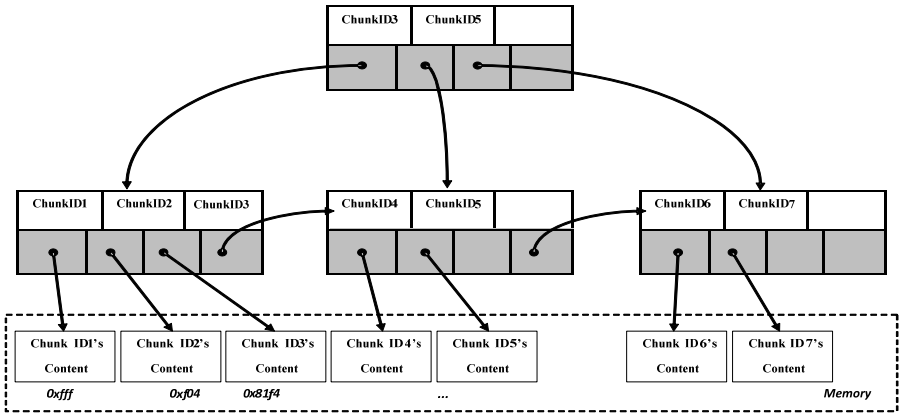

Figure 3. The proposed B+ Tree indexing mechanism using ChunkID as keys

3.2.1. Chunk's Metadata

Chunk metadata is needed for the purpose of reconstruction phase to re-Construct the file or folder that is deduplicated. Figure 4 shows the database design for chunk's metadata.

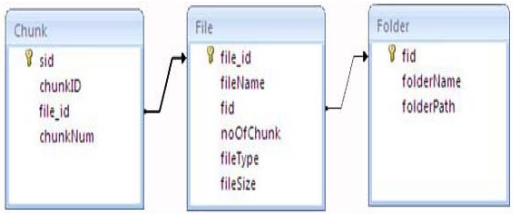

Figure 4. Chunks' Metadata database design

3.3 Storage

In the storage space, the content of the chunk data are stored. And, the metadata including ChunkID based B+ Tree indexing and chunk metadata make interaction with the storage.

3.4 Reconstructor

Once the data, that is, files or folders have deduplicated, the system have Reconstructor to restore the files or folder as original. The flow chart depicts in the Figure 5 shows the process of Reconstructor. It can reconstruct the desired file or folder as the original file or folder. When the reconstructed file has been deleted, the Reconstructor can construct that file again from the metadata and chunks. This means that the system can also make recovery from the accidental delete file from the user.

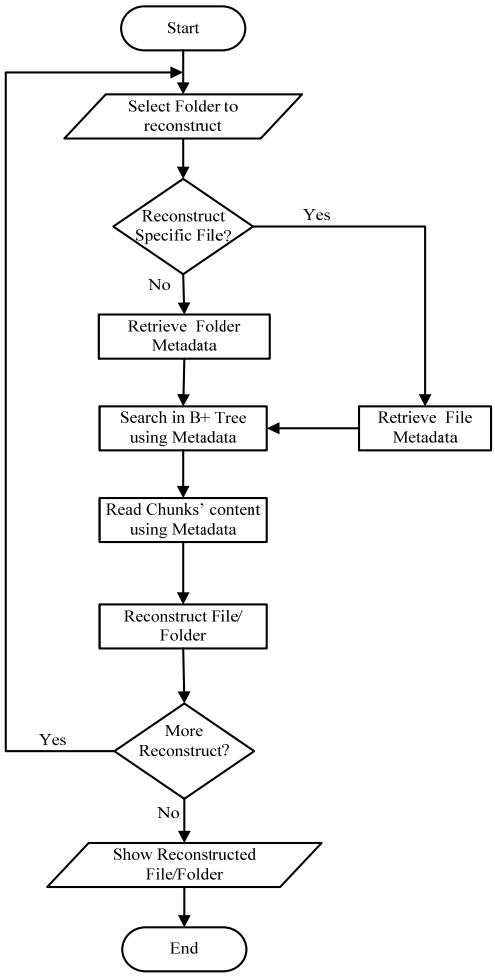

Figure 5. Process of Reconstructor

4. Experimental Results

The experimental setup and comparison results of the proposed system are described in this section.

4.1 Experimental Setup

The experiment is conducted on 2.1 GHz Pentium (R) 4 CPU; 2 GB of RAM (DDR2); 320 GB of hard disk; Microsoft Window XP Professional Version 2002 Service Pack 2; Java SDK 1.6.2.11. And the implementation is made under Eclipse IDE.

Table 1. Datasets List

Tested Datasets	File type	No. of files	File size variation	Dataset's Size (MB)
Dataset_1	PDF	88	1 KB to 8 MB	46.5
Dataset_2	DOC	87	11 KB to 8.23 MB	46.5
Dataset_3	PPT	98	14 KB to 3 MB	46.5
Dataset_4	TXT	132	2 KB to 2 MB	46.5
Dataset_5	HTML	840	1 KB to 837 KB	46.5
Dataset_6	PDF	41	42 KB to 2 MB	20 .5
Dataset_7	DOC	19	11 KB to 2 MB	20 .3
Dataset_8	PPT	62	84 KB to 2 MB	20 .2
Dataset_9	TXT	56	2 KB to 1 MB	20.5
Dataset_10	HTML	300	1 KB to 666 KB	20.6
Dataset_11	PDF	37	42 KB to 1 MB	10 .5
Dataset_12	DOC	29	29 KB to 1 MB	10 .5
Dataset_13	PPT	26	62 KB to 1 MB	10 .5
Dataset_14	TXT	23	3 KB to 2 MB	10.5
Dataset_15	HTML	149	1 KB to 838 MB	10 .5
TestSet_16	Mixed	131	3 KB to 8 MB	46.13

The types of deduplication data (that is, files' types) chosen by users to the deduplication system may be arbitrarily complex in the types of desired files such as portable document format (.pdf), Microsoft Word Document (.doc, .docx), Microsoft PowerPoint Presentation Document (.ppt, .pptx), etc. The proposed system is tested with nineteen Datasets with five file types. The sizes of the files which are involved in the tested DataSets are varied between 1 KB and 8.23 MB and the numbers of files involved in these DataSets are also varied from 23 files to 840 files.

The Table 1 lists the various datasets, with number of files and file size variation that are used as testing data in order to analyze the proposed deduplication system.

As already mentioned, in order to segment the file into various chunks, the TTTD algorithm is used. The proposed system also prepares the thresholds sets to get the reasonable performance in terms of deduplication time and reduce in storage space. The Table 2 lists the tested thresholds sets.

Table 2. Tested Thresholds Sets

Thresholds Set	D	D′	T_{min}	T_{max}
Thresholds Set1 (TS1)	540	270	470	2600
Thresholds Set2 (TS2)	1080	540	940	5200
Thresholds Set3 (TS3)	2160	1080	1880	10400
Thresholds Set4 (TS4)	4320	2160	2760	20800

Where,

D = Main Divisor Threshold Value

D′ = Backup Divisor Threshold Value

T_{min} = Minimum Threshold Value

T_{max} = Maximum Threshold Value

By applying this Thresholds Sets listed in Table 3 to the proposed system with the TestSet_16 (which has 46.13 MB), the resulted number of chunks and the respective processing time are shown in Table 3.

Table 3. Analysis of Chunking with Various Thresholds Sets

Dataset	D	D′	T_{min}	T_{max}	No. of Chunks	Processing Time (ms)
TestSet19	540	270	470	2600	42356	153391
	1080	540	940	5200	21337	129812
	2160	1080	1880	10400	10621	123500
	4320	2160	2760	20800	5477	117578

4.2 Performance Evaluation

When implementing storage of any kind, whether it is important to know just how well that storage performs. A data deduplication ratio over a particular time period is the number of bytes input to a data deduplication process divided by the number of bytes output. The data deduplication ratio relevant in most situations which reflects all of the complementary capacity optimization technologies actually used is calculated as described in Equation (1):

Deduplication Ratio(%) = Bytes In/Bytes Out (1)

In addition, the percentage of the space reduction ratio can calculate as shown in Equation (2):

Space Reduction Ratio(%)
=1-(1/ Deduplication Ratio)*100 (2)

4.3 Experimental Results

The experiments are done on various datasets with four sets of Thresholds in order to obtain the suitable threshold set for the proposed system.

4.3.1 WinZip vs. EIMDD

The performance results are compared with dumb compression or character-based data reduction system, that is, WinZip and intelligent compression, that is, chunk based data reduction system, the proposed system.

The analysis results of the storage consumption, space saving, space reduction percentage as well as data deduplication ratio are shown in Table 4. The testing is done on the TestSet_16 which have 46.13 MB in size with mixed file types.

Table 4. Chunk-based Deduplication Vs. Character-based Deduplication

Type	Description	TestSet_19
WinZip	Actual Usage (MB)	36.30
	Space Saving (MB)	8.83
	Space Reduction (%)	20%
	Deduplication Ratio	1.24 : 1
The Proposed System	Actual Usage (MB)	23.9
	Space Saving (MB)	21.23
	Space Reduction (%)	47%
	Deduplication Ratio	1.89 : 1

The experimental results demonstrated in the followings are resulted from testing on original tested data sets' sizes that are listed in the Table 1. The testing was done on that datasets which the system tried to store three times. Figure 6 represents the comparison results of the WinZip and the proposed system for mixed file types based on TestSet_16.

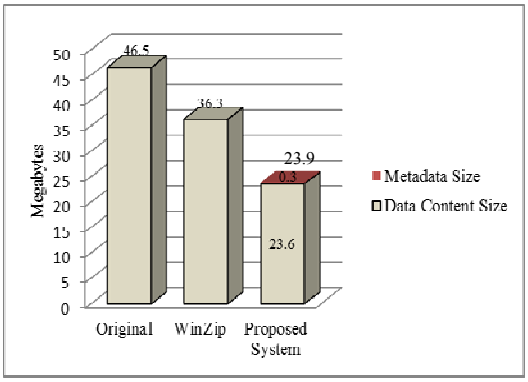

Figure 6. Comparison Results Tested for Mixed File Type

According to the results demonstrated in Figure 6, it can clearly be found that the proposed system uses less storage space than original size as well as the WinZip.

The experimental results from the Figure 7 shows the comparisons result tested for the file type PDF which is DataSet_1 described in Table 1. It is obvious that the proposed system works well with that file type.

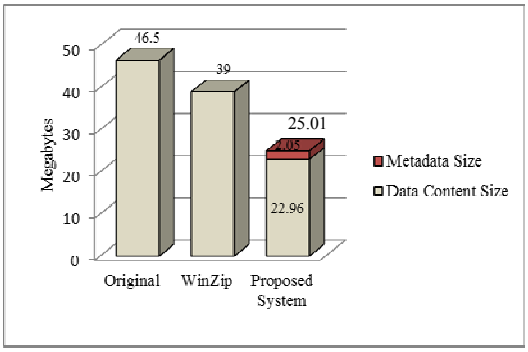

Figure 7. Comparison Results Tested for PDF File Type

Similarly, the results shown in Figure 7 states that the Chunk-based system uses less memory space than Character-based system. This tested was done on the file type of Microsoft Word Document (.doc, .docx).

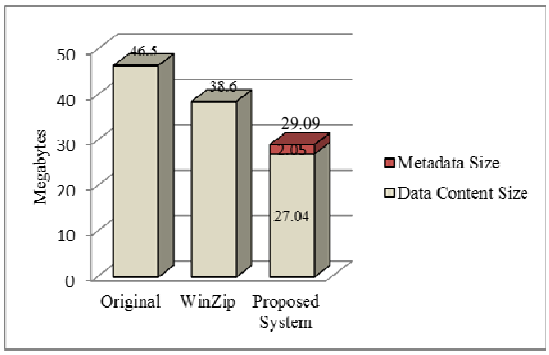

Figure 8. Comparison Results for DOC File Type

The Figure 9 demonstrates the results tested for text file types (.txt, .java, etc.),DataSet_4. According to these results, in text file types, the Character-based system (e.g.WinZip) is more efficient than the chunk-based system in first time store. It can work more efficient in the long sequence of characters. Even it is good in first time store, when the data become redundant; the proposed system is more efficient than the storage consumption for the WinZip.

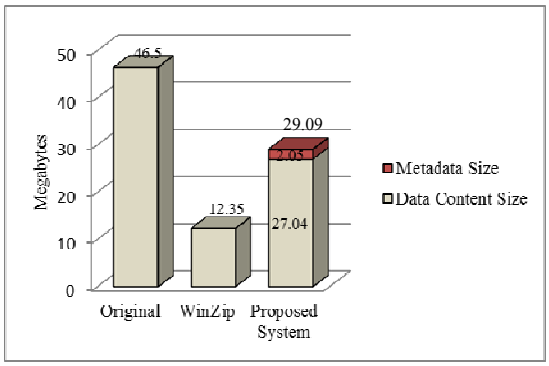

Figure 9. Comparison Results for TXT File Type

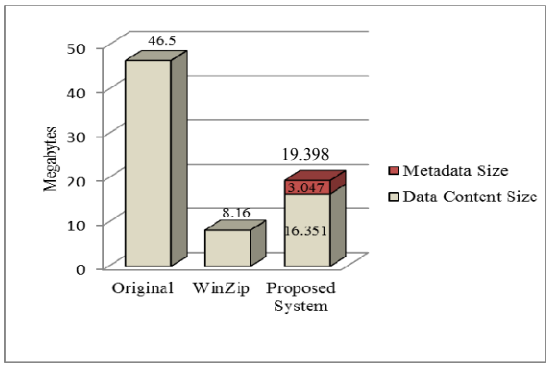

Figure 10. Comparison Results for HTML File Type

The Figure 10 shows the results getting from testing the file type HTML. From these testing results, it can be found that as in TEXT file type testing, the proposed system used more space than WinZip. It is because of the amount metadata size used by the proposed system.

Another testing is done on the file type of Microsoft PowerPoint Presentation files such as .ppt, .pptx. The comparison results between the two systems are illustrated in Figure 11. It uses a slightly less storage amount than WinZip.

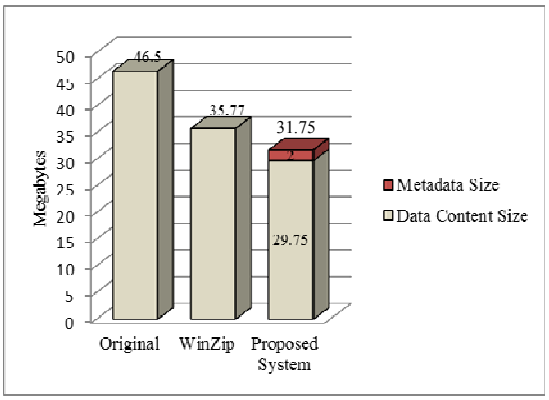

Figure 11. Comparison Results for PPT File Type

As described in the Figure 11, it can clearly see that the proposed system is more efficient in deduplication than traditional system as well as the WinZip.

5. Conclusion

We designed a framework for Efficient Data Deduplication, EIMDD. It can reduce the storage space as its potential purpose. Because of using effective and efficient Hash Algorithm for creating Chunk_ID to check duplicate data, it can be more effective and reliable for security and hash collision. We proposed an efficient indexing mechanism to speed up the searching facility to identify redundant chunks. By using proposed Chunk_ID based B+ tree searching mechanism, significantly reduce the searching time and comparison space.

In this paper we describe the experimental performance results of the proposed system. The initial intention of the proposed system is to reduce storage space for duplicated data in storage area with reasonable processing time. According to the various analyses and testing, it can be found that the proposed system worked well in various file types except for the text oriented files but only at first time storage. It is found that the proposed system can be used as a backup and recovery system. Even the reconstructed file is deleted by the user; the system can restore the specific file

whenever it is needed. Because the system retains the concerned metadata in permanent matter, so that it can be used as a recovery tool as well as it served as the backup of the data files.

References

[1] Alberto H.F. Laender, Altigran Soares da Silva," Learning to deduplicate", *Proc. 8th ACM/IEEE-CS joint conference on Digital libraries (JCDL),* Association for Computing Machinery (ACM) , New York, USA, 2006, pp. 41-50.

[2] Athicha, M., Benjie, C., and David, M., "LBFS: A low-bandwidth network files system". *18th ACM Symposium on Operating Systems Principles (SOSP '01),* Canada, 2001, pp. 174–187.

[3] Bayer .R and MC. Creight, "Organization and Maintenance of Large Ordered Indices", *Acta Informatica, Volume 1*, Springer Berlin/Heidelberg, New York, 1972, pp. 173-189.

[4] Chauanyi, L. et.al, "ADMAD: Application-Driven Metadata Aware De-duplication Archival Storage System", *Fifth IEEE International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI)*, IEEE Computer Society, Los Alamitos, CA, USA, 2008, pp.29-35.

[5] Daniel P. Lopresti, "Models and Algorithms for Duplicate Document Detection", *Proceedings of the Fifth International Conference on Document Analysis and Recognition (ICDAR'99),* IEEE Computer Society, Washington, DC, USA, 1999, pp. 297-300.

[6] Dave Reinsel, "Our Expanding Digital World: Can we contain it? Can we manage it?", *Intelligent Storage Workshop (ISW)*, University of Minnesota, MN, May 2008, pp. 13-14.

[7] Eshghi, K., "A Framework for Analyzing and Improving Content-based Chunking Algorithms", *Technical Report HPL-2005-30(R.1)*, Hewlett-Packard Laboratories, Palo Alto, CA, 2005.

[8] Eshghi, E. et.al., "High Performance Scalable Data Deduplication", Storage Systems Research Center: University of California, 2008.

[9] Jared, D. et.al, "Learning-based Fusion for Data Deduplication", *Seventh International Conference on Machine Learning and Applications* (ICMLA'08), IEEE Computer Society, California, USA, 2008, pp. 66-71.

[10] M. O. Rabin, "Fingerprinting by random polynomials", *Technical Report TR-15-81*, Center for Research in Computing Technology, Harvard University, 1981.

[11] Michael T. Goodrich, *Data Structures and Algorithm in C++*, Wiley Publishing, 2009, pp. 598.

[12] M.Lillibridge et al., "Sparse Indexing, Large Scale, Inline Deduplication Using Sampling and Locality*". 7th USENIX Conference on File and Storage Technologies*, USENIX Association, San Francisco, California, 2009, pp. 111-123.