

**DATA DEDUPLICATION FOR MYANMAR
LANGUAGE STORAGE BY USING SECURE HASH
ALGORITHM**

THAE NU AYE

M.C.Sc.

JANUARY 2023

DATA DEDUPLICATION FOR MYANMAR LANGUAGE STORAGE BY USING SECURE HASH ALGORITHM

By

THAE NU AYE

B.C.Sc.

**A dissertation submitted in partial fulfillment of the
requirements for the degree of**

Master of Computer Science

M.C.Sc.

**University of Computer Studies, Yangon
JANUARY 2023**

ACKNOWLEDGEMENTS

To complete this thesis, many things are needed like my hard work and the supporting of many people who gave a lot of idea to me.

I would like to express very special thanks to **Dr. Mie Mie Khin**, Rector of the University of Computer Studies, Yangon, for her kind permission to submit this dissertation.

I would like to express my deepest gratitude to my thesis supervisor, **Dr. Tin Thein Thwel**, Professor and Head of Faculty of Information Science, University of Computer Studies, Yangon, for her close supervision, proper guidance, valuable suggestions, advice and encouragement during the course of this work.

I would like to thank course coordinators, **Dr. Si Si Mar Win**, Professor, University of Computer Studies, Yangon and **Dr. Tin Zar Thaw**, Professor, University of Computer Studies, Yangon for their superior suggestions and administrative supports during my academic study.

I would like to express my respectful gratitude to **Daw Aye Aye Khine**, Associate Professor and Head of English Department, for her valuable supports from the language point of view and pointed out the correct usage in my dissertation.

I also like to acknowledge my thanks to **all my dear teachers** who taught me from childhood to the master's degree course.

I wish to express my gratitude to **my grandparents, my beloved parents, my elder brother, my younger brother and my younger sister** who supported and encouraged me not to give up my course and their endless love, invaluable support and encouragement to fulfill my wish.

Finally, I am grateful to my colleagues and all my friends for their cooperation and help. I once again extend my sincere thanks to all of them.

STATEMENT OF ORIGINALITY

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

Date

Thae Nu Aye

ABSTRACT

There is a vast amount of duplicated or redundant data in storage systems. The existing data deduplication attempted to reduce the storage spaces in file-level, sub-file-level data storage in terms of byte-level. There is also a need to reduce content level data deduplication, especially in Myanmar language contents. This study aims to deduplicate the data for sentences written in Burmese. The system accepts Myanmar sentences as input and uses Text Splitter to segment the input file into chunks according to the whitespace. Input the separated chunks into the ChunkID generator to generate the ChunkID by applying the Secure Hash Algorithm (SHA1). The system will search for duplicate phrases, and then it will work on reducing those duplicate phrases. The system is implemented with python in Visual Code IDE. According to the tested result, the system can dedupe the duplicated data which are written in Myanmar language with the file type .txt and .docx, especially, it is work well in .txt file for both deduplication and reconstruction process.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	i
STATEMENT OF ORIGINALITY	ii
ABSTRACT	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES	vi
LIST OF TABLES	viii
LIST OF EQUATIONS	ix
 CHAPTER 1 INTRODUCTION	
1.1 Concepts of Data Deduplication	1
1.2 Data Deduplication Techniques	2
1.2.1 Inline Method.....	2
1.2.2 Post-Processing Method.....	2
1.3 Related Work	3
1.4 Objectives of the Thesis	4
1.5 Organization of the Thesis	4
 CHAPTER 2 BACKGROUND THEORY	
2.1 Data Deduplication	5
2.2 Block vs. File-level Data Deduplication.....	6
2.3 File Chunking Approaches	6
2.3.1 Fixed Size Chunking.....	7
2.3.2 Content Defined Chunking	7
2.4 Data Deduplication Benefits	8
2.5 Classification of Deduplication.....	8
2.5.1. Inline Data Deduplication	9

2.5.2	Post process Data Deduplication	10
2.6	File Level Deduplication.....	11
2.7	Sub-File Level Deduplication	11
2.8	Metadata.....	12
2.9	Secure Hash Algorithm (SHA-1).....	12
2.9	Performance Evaluation	16
 CHAPTER 3 SYSTEM ARCHITECTURE		
3.1	Overview of the Proposed System	18
3.2	System Component Descriptions	20
3.3	Data Deduplication Procedure	21
3.4	File Chunker (Text Splitter).....	21
3.5	ChunkID Generator.....	22
3.6	Duplicate Finder.....	23
3.7	Reconstruction	25
3.8	Summary	26
 CHAPTER 4 SYSTEM IMPLEMENTATION		
4.1	Experimental Setup.....	27
4.2	System GUI.....	27
4.3	Performance Results and Discussions	32
4.4	Experimental Findings	38
4.5	Summary	38
 CHAPTER 5 CONCLUSION AND FURTHER EXTENSIONS		
5.1	Advantages and Limitation of the System	40
5.3	Further Extensions	41
AUTHOR’S PUBLICATIONS		42
REFERENCES		43

LIST OF FIGURES

	Page
Figure 2.1 Classification of Data Deduplication.....	9
Figure 2.2 Left and Right shift operation.....	15
Figure 2.3 Data Deduplication Ratio	16
Figure 3.1 Overview of the Proposed System	19
Figure 3.2 Data Deduplication Procedure.....	21
Figure 3.3 Text Splitter	22
Figure 3.4 Flowchart for ChunkID generator	23
Figure 3.5 Duplicate Finder Algorithm	24
Figure 3.6 Flowchart for Duplicate Finder	24
Figure 3.7 Flowchart for Reconstructor process.....	25
Figure 4.1 Main page of Myanmar language Data Deduplication System.....	28
Figure 4.2 File selections to dedupe	28
Figure 4.3 Deduplicated file result, first time store process	29
Figure 4.4 Deduplicated file result, second time store process.....	30
Figure 4.5 File Reconstruction.....	31
Figure 4.6 File selection to reconstruct.....	31
Figure 4.7 File reconstructed result	32
Figure 4.8 Deduplication for 100% duplicate file (.docx).....	33
Figure 4.9 Deduplication for 100% duplicate file (.txt).....	34
Figure 4.10 Deduplication for 50% duplicate file (.docx).....	34
Figure 4.11 Deduplication for 50% duplicate file (.txt).....	35
Figure 4.12 Deduplication for 25% duplicate file (.docx).....	36
Figure 4.13 Deduplication for 25% duplicate file (.txt).....	36
Figure 4.14 Deduplication time comparisons	37
Figure 4.15 Reconstruction time comparisons.....	38

LIST OF TABLES

	Page
Table 2.1 Comparison of x and y	14
Table 2.2 Operation of XOR.....	14

LIST OF EQUATIONS

	Page
Equation 2.1 SHA-1 Iterations 16-79 Formula.....	13
Equation 2.2 Circular Shift Operation Equation.....	14
Equation 2.3 SHA-1 Iterations 80 Formula	15
Equation 2.4 Logical operator Equation	16
Equation 2.5 Deduplication Ratio.....	16
Equation 2.6 Space Reduction Ratio	16

CHAPTER 1

INTRODUCTION

The amount of storage needed by servers grows along with the amount of internet usage, raising the possibility of duplicate data, especially in situation where persistent storage must be kept for a long time. As a result, the storage environment presents an intriguing research topic for data deduplication. Deduplication is the method for reducing storage space by removing redundant data copies kept in various spaces. This method decrease the amount of data stored on each individual device while also decreasing storage costs.

There is currently a significant amount of redundancy and redundant data in storage devices. Data duplication happens both within and between versions of the same file. These vast quantities of duplication of data require more storage space and power, which significantly reduces storage utilization. Data deduplication can get rid of duplicated sections or whole files as well as multiple copies of the same data. Unique data chunks are found, and by replacing redundant chunks with new ones using the right strategy, duplicate data is removed. Users of the Burmese language typically use space however they see fit. In order to reduce data, the Myanmar Language Deduplication in this thesis tries to use String Tokenizer and the Data Deduplication method. It is aimed to remove redundant data in order to reduce storage space.

1.1 Concepts of the Data Deduplication

Data deduplication, also known as intellectual compaction or single-instance storage, tries to search for duplicated content in data. At its most basic, this identifies duplicates of the same file, but deduplication only functions for identical data, so two files that differ by a few bits still need to be considered unique. With today's data deduplication, it is possible to search for repetitive components or bytes much more extensively, which results in much better storage savings. Only the first instance of the data is actually committed to disk when it is decided to move to a backup, archive, or reproduction platform. The only indication for subsequent instances is a tiny stub that makes a reference to the saved iteration. The input data can first be divided into individual chunks, entire files, or streams. This chunking process is important, as it allows for data to be identified by size and shape rather than the actual content.

Data Deduplication techniques that make sure on storage media like disk, flash, or tape, only one distinct instance of the data is kept. Data deduplication can help reduce storage costs and increase efficiency by lowering the amount of storage space required to keep a single instance of data. A pointer to a unique data copy is used to replace redundant blocks of data. Data deduplication can significantly reduce storage capacity by trying to get rid of duplicate data.

1.2 Data Deduplication Techniques

Data Deduplication can be performed either as part of the data processing process (the in-line method) or after the data has been written to the medium (the post-processing method).

1.2.1 Inline Method

The type of data deduplication being used has no bearing on in-line data deduplication. The fact that the data is only processed once and is not processed again after the backup window is a benefit of in-line data deduplication. Since native data is not stored before data deduplication, in-line data deduplication uses less disk space. In-line data deduplication has the drawback of potentially slowing down the backup data stream, depending on how it is implemented. Data deduplication algorithms can be CPU-intensive, and if the index is disk-based, data deduplication may require additional read or write access.

1.2.2 Post-processing Method

When using a post-processing data deduplication technique, data is first backed up, and then the deduplication is carried out after the backup window has ended. The benefit of this approach is that neither the backup window nor the initial backup streams are delayed.

The following are some drawbacks of the post-processing approach:

- Enhanced I/O to the storage device in both directions. Since the data is written during the backup, reads must be performed to check for duplicate data, and if any are found, the pointer(s) must be changed. The overall data deduplication cycle will most likely be longer than if it were done in-line.
- It takes more disk space than an in-line approach since all the data must be stored before deduplication.

It may happen that the data deduplication process is not finished before the beginning of the next backup window if the post-processing data deduplication period goes on for too long.

1.3 Related Work

The researchers M.Lillibridge et al.[14] explained Chunk-lookup disk bottleneck/full chunks indexing encountered in in-line deduplication. This paper proposed to solve using sampling and sparse index and chunk locality. 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.

The researchers May Thu Win et al.[11], Myanmar Natural Language Processing Lab described “Burmese Phrase Segmentation” posed to express how to segment phrases in a Burmese sentence and how to formulate rules. Described how the system can segment sentences into phrases with noun markers, verb markers, zero markers and other techniques in this paper. This has two phases. The first one is encoding phase and the second one is decoding phase. In encoding phase, at first, they collect and normalize raw text from online and offline journals, newspapers and e-books. Next, they train these sentences decoding with CRF++ tool to get Burmese phrase model. The system has been tested by developing a phrase segmentation system using CRF++.

The researcher Walter Santos and Thiago Teixeira [23] explained “A Scalable Parallel Deduplication Algorithm”. This paper proposed to identify the replication of data in database with parallel deduplication algorithm using filter-stream model. They considered for databases and not for sub-file level of the file which have their respective secure hash code.

1.4 Objectives of the Thesis

The major aim of this thesis is to develop a completely automated data deduplication system that requires less indexing time and less duplicate data storage space and other major objectives are as follows:

- To remove redundant data in order to reduce storage space.
- To identify the same data by using the advantage of cryptographic technology.
- To obtain the efficient mechanism for searching identical data in the existing storage for Myanmar language.

1.5 Organization of the Thesis

The thesis is divided into five chapters and is structured as follows:

Chapter 1 introduces the purpose of this thesis as well as the principles of data deduplication, methodologies, and related work.

Chapter 2 explains background theory for data deduplication and Secure Hash Algorithm. The Evaluation method used in this system is also explained in this section.

Chapter 3 describes the design of the Data Deduplication for Myanmar Language Storage using Secure Hash Algorithm. This section includes a full explanation of a step-by-step procedure and a system design diagram. It is followed by a description of the suggested algorithms that are a part of this study.

Chapter 4 provides a thorough case and the benefits of using Python. The outcomes of the research, which have collected for the thesis, are also provided. System implementations are shown with the relevant figures in this section.

Last but not least, in **Chapter 5**, it concludes with the proposed system and discusses its limits as well as suggested directions for further study.

CHAPTER 2

BACKGROUND THEORY

This chapter intends to point the background theory applied in this research work. It explains data deduplication, secure hash algorithm and evaluation methods used in this work.

Data Deduplication is a process that eliminates excessive copies of data and significantly decreases storage capacity requirements. Deduplication can be run as an inline process as the data is being written into the storage system and/or as a background process to eliminate duplicates after the data is written to disk. Deduplication operates eliminating duplicate data blocks and storing only unique data blocks. In this proposed system, Myanmar Language Deduplication was emphasized.

Data Deduplication working mechanism is as follow: Deduplication segments an incoming data stream, uniquely identifies data segments, and then compares the segments to previously stored data. If the segment is unique, it's stored on disk. However, if an incoming data segment is a duplicate of what has already been stored, a reference is created to it and the segment is not stored again.

2.1 Data Deduplication

An advanced type of data compression is data deduplication. The application of this technology has the potential to revolutionize data protection, speeding up disk-based methods for backup and recovery as well. Data deduplication is essentially a way to save storage space, to put it another way. It functions by removing redundant data and making sure that only one distinct instance of the data is really kept on the storage medium, such as a disk or tape. A pointer to the original copy of the data is used to replace redundant data. Data deduplication, also known as single-instance storage or efficient compression, is frequently used in combination with other data reduction techniques. Traditional compression has been used to reduce big or repetitive data sets for almost three decades now by using mathematical techniques to the data. Data deduplication does something similar by finding redundancies within the data and deleting them, allowing only one unique instance of a piece of data to remain.

2.2 Block vs. File-level Data Deduplication

Data deduplication is frequently carried out either at the block or file level. Duplicate files can be removed using file deduplication, although this approach is not particularly efficient. A file that has to be archived or backed up is compared with copies that already exist as part of data deduplication at the file level. By contrasting its characteristics with an index, this is achieved. Only a link to the existing file is stored unless the file is unique, in which case it is saved and the index is modified. As a result, only one duplicate of the file is saved, and any other copies are swapped out for stubs that refer to the original.

Block-level Deduplication means that a file's unique iterations of each block are sought for and saved via deduplication. The blocks are divided into identical-length sections. Each block of data is processed using a hash algorithm like MD5 or SHA-1. By using this technique, each component is given a special number that is then entered into an index. When a file is updated, just the modified data is saved, even if the content or presentation has only undergone minor changes. No new file is created as a result of the changes. This behavior makes block deduplication considerably more efficient. On the other side, block deduplication uses a significantly bigger index and more processing resources to track each individual file.

Variable-length Deduplication is a method for dividing a file system into pieces of different sizes, offering greater data reduction ratios than fixed-length blocks. Cons are that it produces more information and moves more slowly. With deduplication, hash collisions may be a problem. A piece of data is assigned a hash number, which is then compared to the index of other hash numbers that already exist. If the hash value is already in the index, which has a better data reduction ratio than fixed-length blocks, the data is considered redundant and doesn't need to be saved again. Cons are that it produces more information and moves more slowly. With deduplication, hash collisions may be a problem. A hash number is compared to the index of any existing hash numbers.

2.3 File Chunking Approaches

Deduplication techniques separate an input object (or stream of files) into smaller pieces known as "chunks," such as blocks or segments, and store only the distinct chunks. Traditionally, there are several ways to chunk an object, including

content-dependent fingerprinting, fixed size chunks (where each chunk is the same length), and others. One drawback of these chunking techniques is that deduplication performance (compression ratio) is better with smaller chunk sizes, regardless of the chunking techniques used. The costs of metadata depend on how many chunks are generated for a file, regardless of the metadata scheme used. The smaller the chunk size, the more effective the data deduplication. It indexes the chunks and raises the overhead of those chunks' hashes.

2.3.1 Fixed Size Chunking

A file is divided into fixed size units, such as 8 KB blocks, in fixed size chunking. It is easy to use, quick, and inexpensive to compute. Chunking moves at the same speed as IO and is not constrained by the CPU. For backup applications and massive file systems, fixed-size chunking has been used in a number of subsequent works. It has one flaw, though. Even though the majority of the file's content is still present, the fixed size chunking may produce a different set of chunks when a small amount of content is added to or removed from the original file. Even so, it is helpful when files are not frequently updated.

Fixed blocks may be 8 KB in size or even 64 KB; the difference is that the likelihood that a chunk is redundant increases with decreasing size. As a result, reductions will be even greater as less data is stored. The only drawback is that, if a file is modified and the deduplication tool uses the same fixed blocks from the previous inspection, it might fail to identify redundant segments because, as blocks are changed in a file, they change downstream from the transformation, offsetting the other comparisons.

2.3.2 Content Defined Chunking

Data deduplication, an effective method of data reduction, has grown in popularity and attention among large-scale storage systems as a result of the exponential growth of digital data. The redundant data is removed at the file or chunk level, and duplicate contents are recognized by their cryptographically secure hash signatures (e.g., SHA1 fingerprint). Because it locates and eliminates redundancy at a finer granularity, chunk-level deduplication is generally more popular than file-level deduplication. The simplest chunking method for chunk-level deduplication involves dividing the file or data stream into equal, fixed-size chunks, also known as "fixed-

size chunking" (FSC). Approaches based on Content-Defined Chunking (CDC) are suggested as a solution to the FSC approach's boundary-shift issue. To be more precise, CDC establishes chunk boundaries based on the data stream's byte contents.

Content-Defined Chunking typically consists of two distinct and sequential steps:

- (1) Hashing, which produces fingerprints of the data contents, and
- (2) Hash judgment, which compares fingerprints to a preset value to determine and declare chunk cut-points. The proposed system used content-defined chunking.

2.4 Data Deduplication Benefits

Deduplication systems have resource-saving advantages. Savings on various levels are possible. The main advantage is a significant reduction in the amount of disk space needed to store a certain amount of data.

Deduplication also makes it possible to enhance service-level agreements for recovery, which is another major benefit. In order to lessen the impact of backups on production activities, backups to disk can increase backup windows and enable production resources to quickly resume normal operations.

Deduplication can therefore reduce restore times because it increases the likelihood that a restore request can be fulfilled from a disk copy rather than a slower access medium like tape or optical. This is because the larger a disk's capacity to store data is, the more likely it is that a restore request can be satisfied from a disk copy.

2.5 Classification of Deduplication

According to the information in Figure 2.1, which illustrates a straightforward relationship between these elements, data deduplication can be categorized technologically.

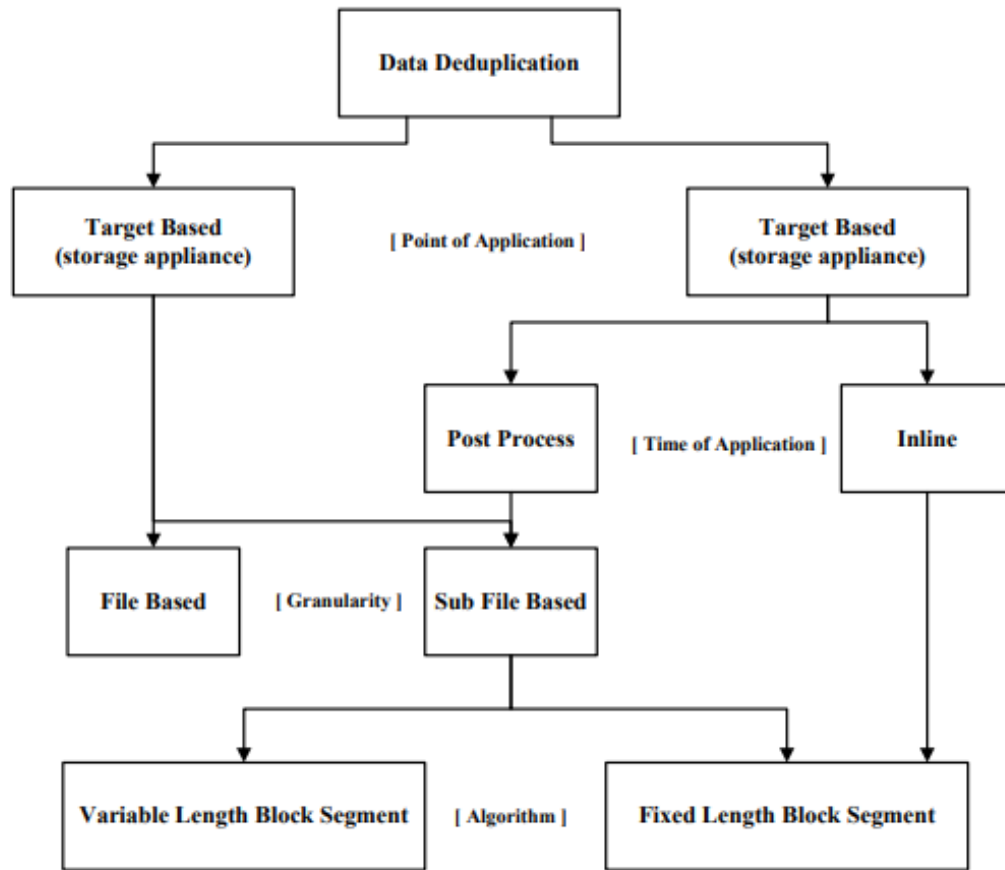


Figure 2.1 Classification of Data Deduplication

2.5.1 Inline Data Deduplication

In inline systems, data is duplicated as it enters the system and before it is written to disk. It is quicker, easier, and safer than post-process deduplication systems, and it can store data on disk. Due to the slower performance, the increased risk, and the management complexity, the storing data to disk first and deduplication it later as a post-process loads backup and recovery.

As part of a larger stream of data from a backup, a block of data enters the appliance. The appliance works its magic to determine whether it has seen that block before while it is in RAM. If it has already seen it, it places a pointer somewhere indicating that it has. It writes the new block of data if it has never seen the previous block of data. The task is finished. With this approach, the bulk of the labor-intensive work can be completed in RAM, reducing I/O overhead. The hash lookup is the only disk operation that is always performed. Some systems also need to read the block, and in order to do so, the new block appears to match.

2.5.2 Post process Data Deduplication

It is a procedure where the data is first gathered in a holding area on the disk and then deduplicated in batch mode. The original data must be written, read, checked for redundancies, and if any are found, one or more pointers must be written. In other words, it can be explained as follows:

An entire block of data is written to disk when it enters the appliance as part of a larger stream of data from a backup. Then, a different process reads the block of data and performs its magic to determine if it has ever seen that block of data before (possibly running sequentially and possibly from another appliance accessing the same file). If so, it replaces the redundant block of data with a pointer instead of deleting it. It makes no difference if they have not seen it. Compared to the inline method, this one requires a lot more I/O—roughly 40. It completely writes all fresh blocks to disk. Then it reads all the new blocks from the disk in their entirety.

Post-process deduplication involves first storing new data on the storage medium, followed by an analysis step to check for duplicates. The advantage is that data can be stored right away without having to wait for the hash calculations and lookups to finish, protecting store performance. Users of implementations with policy-based operation may be able to process files according to type and location while deferring optimization on "active" files. One possible drawback is that users might unnecessarily store duplicate data for a brief period of time, which is problematic if the storage system is almost at capacity. The inability to accurately predict when a process will be finished is probably the biggest problem in the real world.

Post-processing has the following benefits:

- i. There are no worries about incoming backup speed slowing down;
- ii. Dedupe implementation can be staggered;
- iii. Users can copy last night's backups in their original format; and
- iv. Post-processing permits a forward-referencing approach (if desired).

The following are some drawbacks of post-processing: It requires more configuration than an inline approach due to the following reasons:

- i. It requires a lot more I/O work;
- ii. It requires the landing zone disk;

- iii. Data must be processed twice during ingesting and subsequent deduplication; and
- iv. It requires additional disk space to store data that has not yet been deduplicated.

In contrast to the inline method, which can only apply one process per backup stream, users can use more parallel processes (and processors) to solve the problem. This strategy has two drawbacks. Prior to deduplication, the backup must first be stored, necessitating the availability of temporary storage space. Second, there is a requirement for a lot of disk bandwidth because backup files must be read, duplicated, and the new deduplicated backup written to disk. Inline data deduplication is used in the proposed system.

2.6 File Level Deduplication

File-level data deduplication, also known as Single-Instance Storage (SIS), compares a file that needs to be deduplicated with those that have already been stored by comparing its attributes to an index. Only a pointer to the existing file is stored if the file is not unique; otherwise, it is stored and the index is updated. As a result, only one copy of the file is saved, and any additional copies are swapped out for "stubs" that point to the original file. Read-only data can benefit from file level deduplication, but read-write data necessitates greater granularity. It is quick and easy, but it does not deal with the issue of duplicate content found in various files.

2.7 Sub-File Level Deduplication

The file is typically divided into chunks or blocks, as suggested by their names, and each one is checked for duplication with respect to previously saved data. The most common method for finding duplicates is to give a piece of data a unique ID or "fingerprint," such as by using a hash algorithm, which creates a unique ID. A central index is then used to compare the unique ID. The data segment has already been processed and stored if the ID is present. So, all that needs to be saved is a pointer to the previously saved data. The block is unique if the ID is brand-new. The unique chunk is stored, and the unique ID is added to the index.

2.8 Metadata

There are two different types of data produced when storing data in a deduplication system. First, there are protected objects like Word documents, databases, and Exchange message stores. The proposed system will deduplicate these files, and for the simplicity, it will refer to this as "actual storage." Metadata is the second category of data that is produced. The deduplication software uses this data to identify redundancies and may rehydrate data in the event of restoration. These two categories of data are essential and frequently necessary for both writing data to the system and possibly reading data.

Metadata are facts about other facts. The term "metadata" in this system refers to the details about files and the chunks that make up each file. As a result, it can be referred to as chunk metadata. Understanding how the storage system is set up to safeguard this data is crucial. Since running out of metadata space is essentially the same as running out of object space, allocating storage for metadata can be crucial. As a result, metadata is a crucial component of the deduplication process, and its loss or corruption can cause serious problems. Key elements of every deduplication algorithm are the creation, management, and handling of metadata.

2.9 Secure Hash Algorithm (SHA-1)

Another important use of cryptography is protecting data integrity by using hashes. A hash is a checksum that is unique to a specific file or piece of data. A hash value is used to verify that a file has not been modified after the hash was generated. Creating a hash is a one-way operation. Hashes are often used to enable passwords (keys) to be verified without storing the password (key) itself. After the hash of the password (key) has been stored, the application can verify the password (key) by calculating the hash of the password (key) and comparing it with the stored hash. The two hash values will match if the user has provided the same password (key).

The National Institute of Standards and Technology and the National Security Agency created the Secure Hash Algorithm 1, a message digest algorithm. The 160-bit (20-byte) message digest generated by SHA1 is used to create unforgivable digital signatures. The algorithm is slower than MD5, but the message digest is bigger, making it more resistant to brute force attacks, which attempt to generate the same message, digest by picking messages at random. As one-way hash functions, they are

unable to calculate the original data. Additionally, they can serve as signatures. The proposed system makes use of this function to locate duplicate chunks among the various chunks of different files.

Suppose the message ‘abc’ was to be encoded using SHA-1, with the message ‘abc’ in binary being

01100001 01100010 01100011

And that in hex being

616263

Step (1) the first step is to initialize five random strings of hex characters that will serve as part of the hash function (shown in hex):

$H_0 = 67DE2A01$

$H_1 = BB03E28C$

$H_2 = 011EF1DC$

$H_3 = 9293E9E2$

$H_4 = CDEF23A9$

Step (2) the message is then padded by appending a 1, followed by enough 0s until the message is 448 bits. The length of the message represented by 64 bits is then added to the end, producing a message that is 512 bits long.

Step (3) the padded input obtained above, M is then divided into 512-bit chunks, and each chunk is further divided into sixteen 32-bit words, $W_0 \dots W_{15}$. In the case of ‘abc’, there’s only one chunk, as the message is less than 512-bits total.

Step (4) for each chunk, begin the 80 iterations, i necessary for hashing (80 is the determined number for SHA-1), and execute the following steps on each chunk, M_n : For iterations 16 through 79, where $16 \leq i \leq 79$, perform the following operation:

$$W(i) = S^1((W(i-3) \oplus W(i-8) \oplus W(i-14) \oplus W(i-16))), \quad (2.1)$$

Where XOR, or \oplus , is represented by the following comparison of inputs x and y .

Table 2.1 Comparison of x and y

X	y	Output
0	0	0
1	0	1
0	1	1
1	1	0

For example, when i is 16, the words chosen are $W(13)$, $W(8)$, $W(2)$, $W(0)$ and the output is a new word, $W(16)$, so performing the XOR, or \oplus , operation on those words will give this:

Table 2.2 Operation of XOR

$W(0)$	01100001 01100010 01100011 10000000
$W(2)$	00000000 00000000 00000000 00000000
$W(8)$	00000000 00000000 00000000 00000000
$W(13)$	00000000 00000000 00000000 00000000
	\oplus
$W(16)$	01100001 01100010 01100011 10000000

Circular Shift Operation

Now, the circular shift operation $S^n(X)$ the word X by n bits, n being an integer between 0 and 32, is defined by:

$$S^n(X) = (X \ll n) \text{ OR } (X \gg 32 - n), \quad (2.2)$$

where, $X \ll n$ is the **left-shift** operation, obtained by discarding the leftmost n bits of X and padding the result with n zeroes on the right.

$X \gg 32-n$ is the **right-shift** operation obtained by discarding the rightmost n bits of X and padding the result with n zeroes on the left. Thus $S^n(X)$ is equivalent to a circular shift of X by n positions, and in this case the circular left-shift is used. The operation of left and right shift is as shown in the following Figure 2.2.

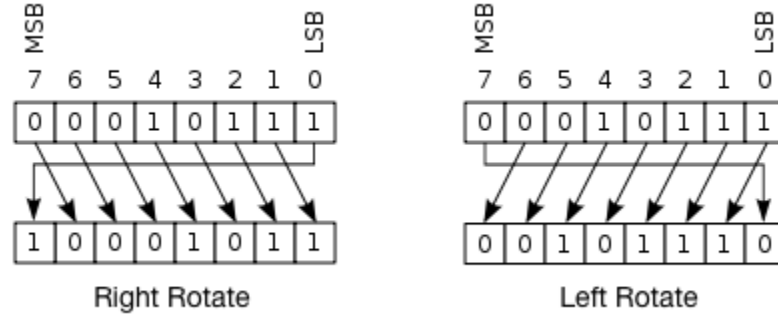


Figure 2.2 Left and Right shift operation

So, a left shift $S^n(W(i))$, where $W(i)$ is 10010, 10010, would produce 01001, as the rightmost bit 0 is shifted to the left side of the string. Therefore, $W(16)$ would end up being:

11000010 11000100 11000111 00000000

Step (5) now; store the hash values defined in step 1 in the following variables:

$$A = H_0$$

$$B = H_1$$

$$C = H_2$$

$$D = H_3$$

$$E = H_4$$

Step (6) For 80 iterations, where $0 \leq i \leq 79$, compute

$$TEMP = S^n * (A) + f(i; B, C, D) + E + W(i) + K(i) \quad (2.3)$$

See below for details on the logical function, f and on the values of $K(i)$. Reassign the following variables:

$$E = D$$

$$D = C$$

$$C = S^{30}(B)$$

$$B = A$$

$$A = TEMP$$

Step (7) Store the result of the chunk's hash to the overall hash value of all chunks, as show below, and proceed to execute the next chunk:

$$H_0 = H_0 + A$$

$$H_1 = H_1 + B$$

$$H_2 = H_2 + C$$

$$H_3 = H_3 + D$$

$$H_4 = H_4 + E$$

Step (8) as a final step, when all the chunks have been processed, the message digest is represented as the 160 bits string comprised of the OR logical operator, \vee , of the 5 hash values.

$$HH = S^{128}(H_0) \vee S^{86}(H_1) \vee S^{64}(H_2) \vee S^{32}(H_3) \vee H_4 \quad (2.4)$$

So, the string 'abc' becomes represented by a hash value to **'a9993e364706816aba3e26727850c26c9cd0d89d'**.

If the string changed to 'abc', for instance, the hashed value would be drastically different so attackers cannot tell that it is similar to the original message. The hash value for 'abc' is **'81fe8bfe87576c3ecb22426f8e57847382917acf'**.

2.10 Performance Evaluation

Understanding how well a storage system performs is essential until placing it into use. The ratio of input bytes to output bytes in a data deduplication process is known as the data deduplication ratio over a given time period. The formula for calculating the data deduplication ratio, which accounts for all complementary capacity optimization technologies actually employed, is given in Equation (2.5) and (2.6):

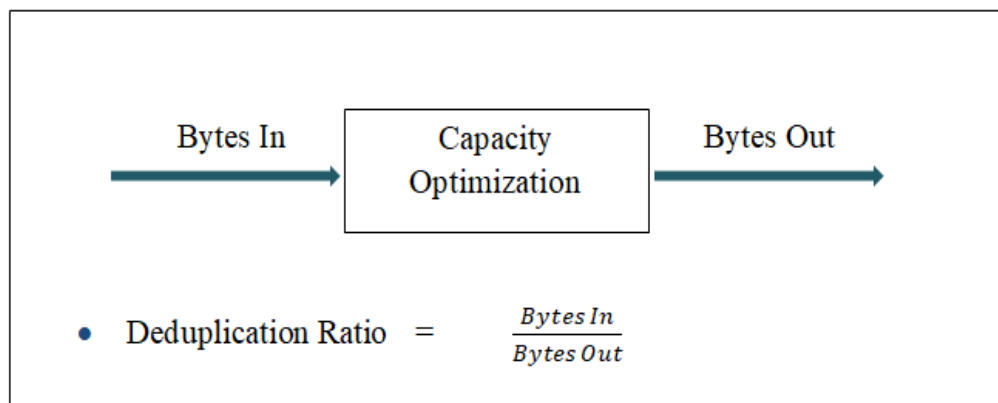


Figure 2.3 Data Deduplication Ratio

$$\text{Deduplication Ratio (\%)} = \text{Bytes In/Bytes Out} \quad (2.5)$$

$$\text{Space Reduction Ratio (\%)} = [1 - (1/\text{Deduplication Ratio})] * 100 \quad (2.6)$$

Figure 2.3 shows the space reduction ratio relevant in most situations, which reflects all of the complementary capacity optimization technologies actually used. The chunkID's in our implementation are generated using the SHA1 hash algorithm. Internally, the data is saved as files. A data deduplication ratio over a particular time period is the number of bytes input to a process divided by its output. This ratio provides an indication of how well the deduplication process has worked and thus gives us a valuable insight into how much disk space those are able to save. With the use of the SHA1 hash algorithm to generate chunkID's, it can achieve an efficient data deduplication ratio over any given period. By taking advantage of the fact that data is typically composed of multiple copies and that each file contains similar information, the system can use deduplication technology to detect and store only one copy of each block of data.

CHAPTER 3

SYSTEM ARCHITECTURE

Data deduplication is the method of storing and/or sending only unique data after examining a set of data or byte stream at the sub-file level. The approach taken to assess, identify, track, and prevent duplication is the fundamental basis for understanding the term the sub-file level deduplication. The deduplication procedure entails updating tracking data, storing and/or sending new and unique data and ignoring any duplicate data.

Data is compressed by being encoded in order to use less storage space. While loss data compression methods permanently discard some of the original data, lossless data compression techniques enable exact reconstruction of the original data from the compressed data. Data deduplication is a process that gets rid of extra copies of data and drastically reduces the amount of storage space needed. Deduplication can be implemented as a background process to remove duplicates after the data has been written to disk or as an inline process to remove duplicates as the data is being written into the storage system. By removing duplicate data blocks and storing only unique data blocks, deduplication works. In this proposed system, Myanmar language deduplication was emphasized in this work. Data deduplication's mechanism of operation: Deduplication divides an incoming data stream into distinct data segments, then uniquely identifies the segments and compares them to previously saved data. The segment is saved on disk if it is distinct. However, a reference is made to the incoming data segment, and it isn't stored if the segment is a duplicate of what has already been stored.

3.1 Overview of the Proposed System

Figure 3.1 illustrates the layout of the proposed system. The following are the principal parts of the system: The text splitter, ChunkID generator, duplicate finder, metadata, and storage are the system's major components. Txt and docx are two possible file types for input in the Burmese language. File Chunker splits input files into chunks of variable length. The File clunker's output chunks are used by the ChunkID Generator to generate ChunkID by applying the secure hash function SHA1, which is renowned for its resistance to hash collisions. Duplicate Finder uses the

chunk ID to determine whether or not that chunk ID is already present in the metadata.

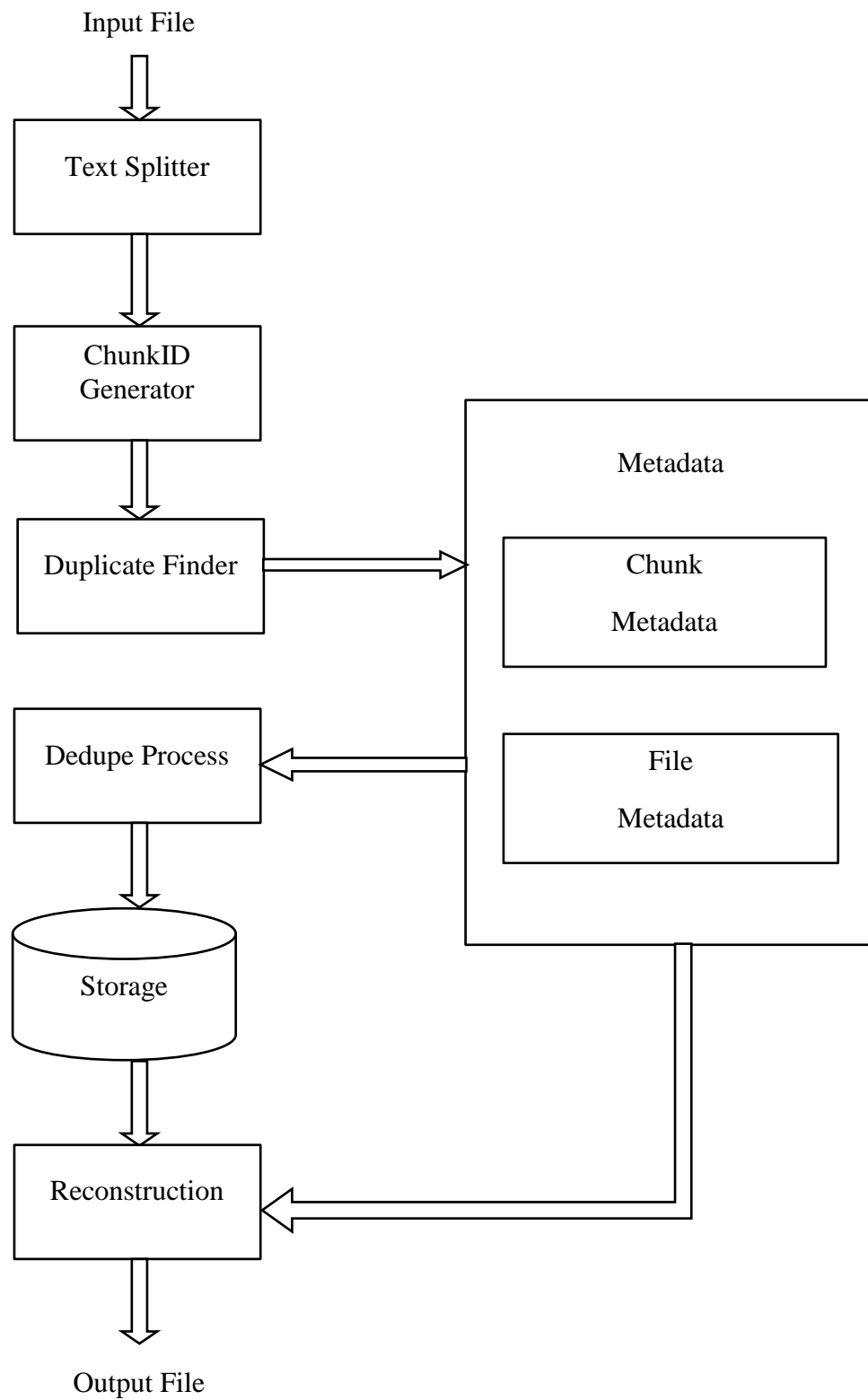


Figure 3.1 Overview of the Proposed System

The metadata maintains the hash code of the already stored file information, including ChunkID. Filename, chunk serial number, Chunk_ID, and Number_of_Chunk are all part of the file information. The storage space contains the contents of the data chunk. After deduplication the data or files, the system uses Reconstructor to restore the files or folders to their original state. It can reconstruct the desired file from the original file. When the reconstructed file has been deleted, the Reconstructor can reconstruct it again from the metadata and chunks. This means that the system can also recover a file that the user has accidentally deleted.

3.2 System Components Description

- Input File
 - Enter Burmese sentences file as input.
- Text Splitter
 - In order to segment the input file as the chunks according to the whitespace, the split () method that splits a string into a list is used.
- ChunkID Generator
 - To generate the ChunkID, the ChunkID Generator uses SHA1 which produces 160 bits signature for each chunk.
- Duplicate Finder
 - The Duplicate Finder finds the duplicate ChunkID in the existing hash code with the incoming ChunkID.
- Metadata
 - The metadata maintain the hash code of the already stored files information including ChunkID, file information.
 - File information includes Filename, Chunk serial number, Chunk_ID, and Number_of_Chunk.
- Storage
 - In the storage space, the content of the chunk data is stored.
- Reconstruction
 - After the input files are deduplicated, it needs to reconstruct in order to obtain the original input files.

3.3 Data Deduplication Procedure

The efficient data deduplication procedure for the proposed data deduplication process contains six steps. They are shown in the Figure 3.2.

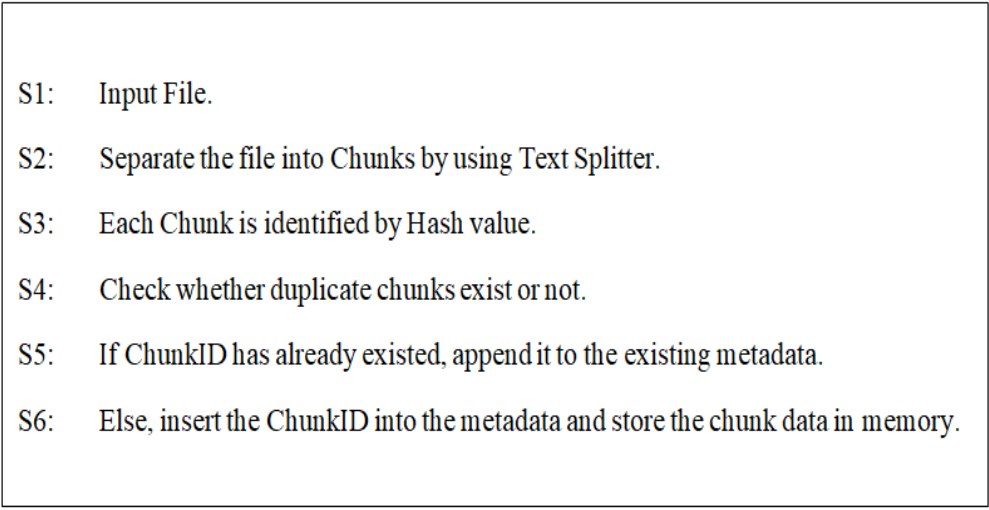
- 
- S1: Input File.
 - S2: Separate the file into Chunks by using Text Splitter.
 - S3: Each Chunk is identified by Hash value.
 - S4: Check whether duplicate chunks exist or not.
 - S5: If ChunkID has already existed, append it to the existing metadata.
 - S6: Else, insert the ChunkID into the metadata and store the chunk data in memory.

Figure 3.2 Data Deduplication Procedure

As illustrated in Figure 3.2, after accepting the input files that written in Myanmar language in S1, these files' content are separated into chunks by using Text Splitter in S2. After that, the resulted chunks are applied by SHA1 in order to generate hash code for each chunk, which is called ChunkID, in S3. Then in S4, the ChunkID is used to check whether the duplicated chunks exist or not. If the ChunkID has already existed, then append it to the existing metadata in S5. If the ChunkID does not exist, then insert that ChunkID into the metadata and store the chunk data in secondary storage were done in S6.

3.4 File Chunker (Text Splitter)

A file is split into segments or chunks by the process of "chunking". Python's split method can be used to break down text and make it easier for computers to understand. Almost every word written in Myanmar language documents is already formatted and segmented with spaces. This process, which makes text analysis easier, is called tokenization. In this system, a simple split () method that splits a string into a list is used.

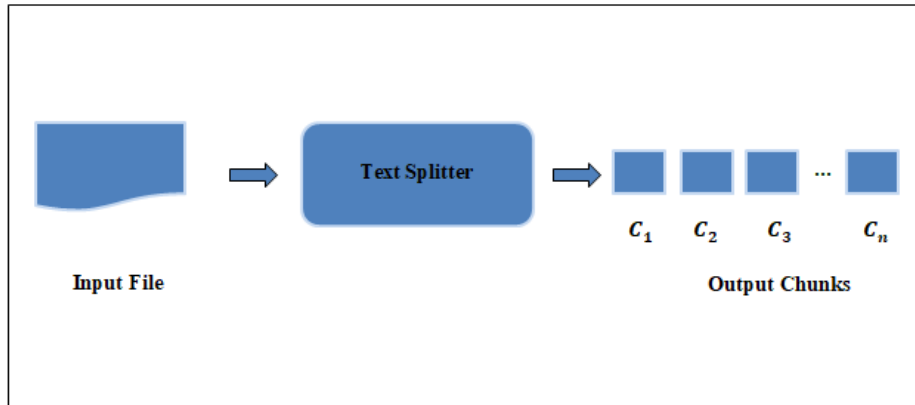


Figure 3.3 Text Splitter

In the above Figure 3.3, a file is segmented or chunked deterministically by text splitter. Smaller segments produce better deduplication, but they produce more chunks and metadata because there are more of them. On the other hand, large chunks can reduce the chance of discovering duplicate data.

3.5 ChunkID Generator

A hashing algorithm can be used to find the identical chunks and produce the Chunk ID. The Chunk ID Generator in this system employs SHA1, which generates a 160-bit signature for each chunk and also features a collision-resistant feature. It can lessen the possibility of identical information being found in the file. The flowchart for the ChunkID Generator procedure is shown in Figure 3.4.

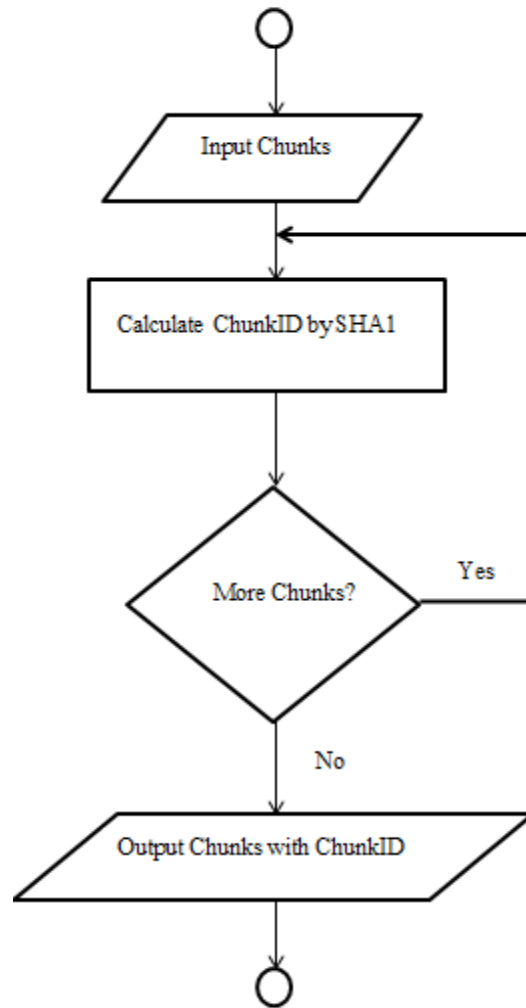


Figure 3.4 Flowchart for ChunkID Generator

3.6 Duplicate Finder

After making each chunk respectively, ChunkIDs generated from ChunkID Generator are used by Duplicate Finder for further processes. Firstly, the Duplicate Finder checks the ChunkID that is already presented or not in ChunkLib. If there is, it gets Address of that chunkID's content and updates the metadata with ChunkIdIndex and ChunkIdMetadata. If not, it updates the Chunklib with new ChunkID and the metadata concerned with the chunk and then store that new Chunk. The algorithm and flowchart for the Duplicate Finder is demonstrated in Figure 3.5 and Figure 3.6.

Duplicate Finder (ChunkID, ChunkIDMetadata, Chunk)

Begin

found \leftarrow searchInchunklib (ChunkID)

if found in Chunklib then

 get index of that chunkID's content

 updateFileMetadata (chunkIdIndex, ChunkIDMetadata)

else

 updateChunklib (ChunkID)

 updateFileMetadata (chunkIdIndex, ChunkIDMetadata)

 store (chunk)

End

Figure 3.5 Duplicate Finder Algorithm

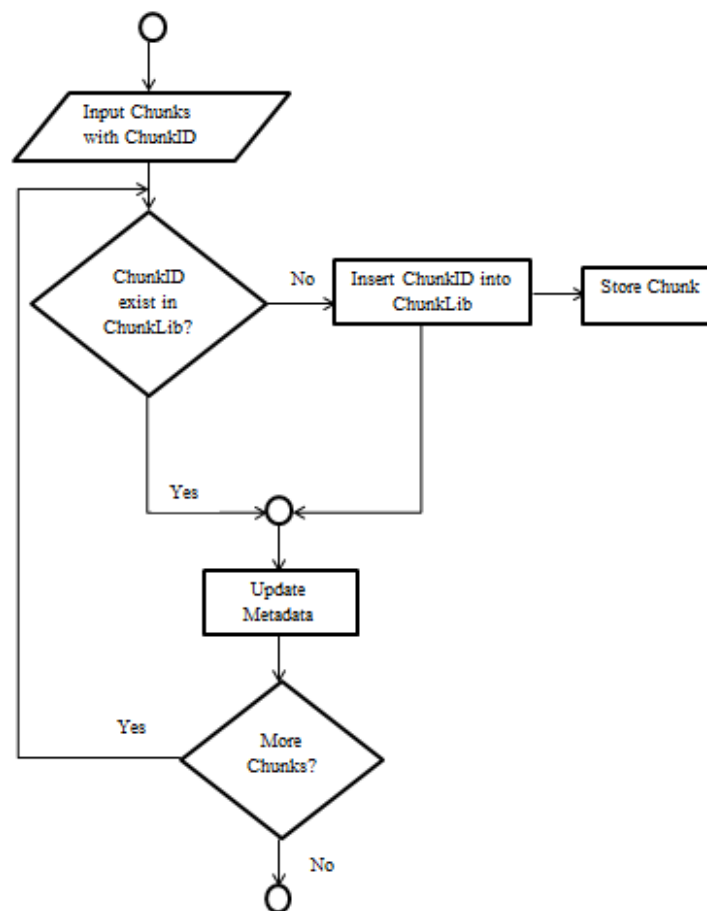


Figure 3.6 Flowchart for Duplicate Finder

3.7 Reconstruction

Once the data, that is, files have deduplicated, the system uses Reconstructor to restore the files or folder as original. It can reconstruct the desired file as the original file. 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 as shown in Figure 3.7.

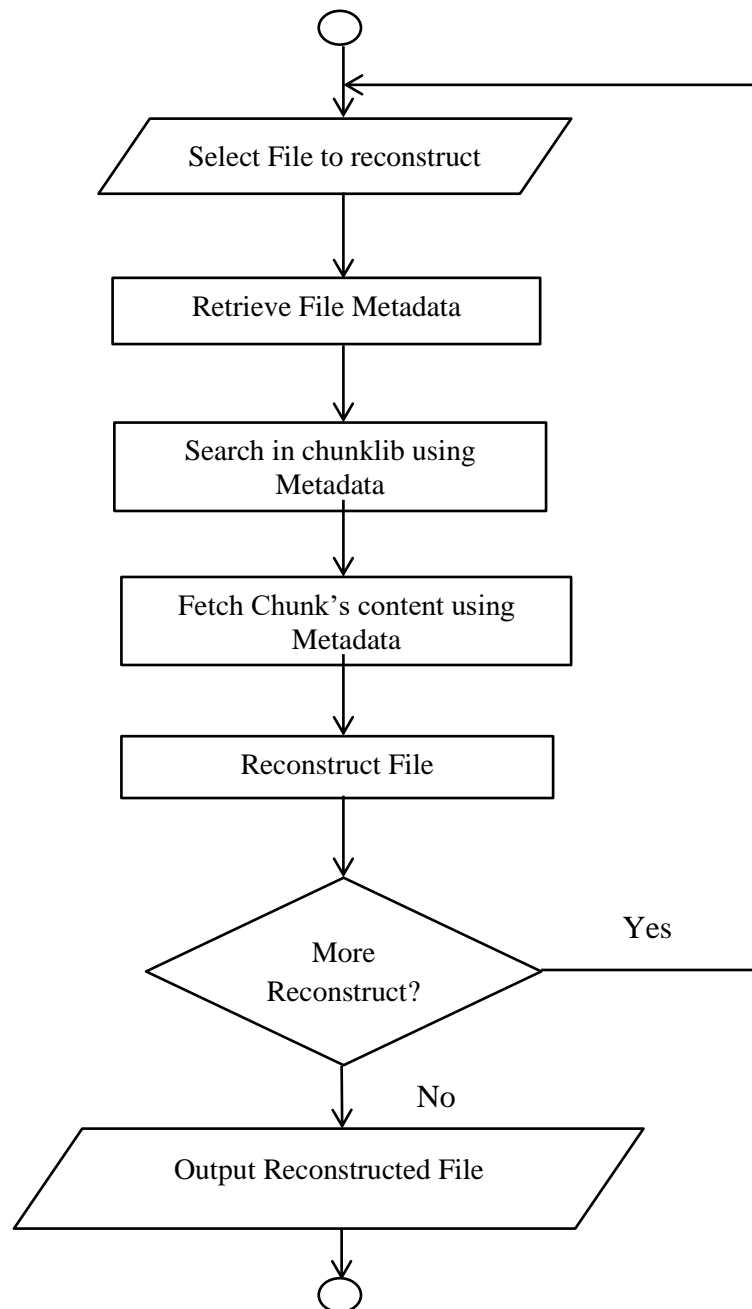


Figure 3.7 Flowchart for Reconstructor Process

3.8 Summary

In this chapter, the detailed descriptions and explanation for the design and implementation of the proposed system are explained. Firstly, it mentioned the overall architecture of the proposed framework. The flow diagrams for the process of each component in proposed framework have been described. The processes for each component are explained. In addition, for checking the integrity of the reconstructed files from the system, the detailed implementation for the integrity checker and the process of the integrity checker have also been explained. And the discussion concerned with the results and performance of the proposed system appears in the next chapter.

CHAPTER 4

SYSTEM IMPLEMENTATION

In this chapter, the implementation of this system is presented with step by step processes. For improving the user understandability when presenting this system, this chapter collects and shows the results with full screen shots. After reviewing all sections in this chapter, the user will be able to know how the system setup and how to process the analysis task.

4.1 Experimental Setup

The experiment is conducted on:

- Hardware configurations
 - CPU: at least Core i3 M 560 @ 2.67 GHz 40
 - Memory (RAM): at least 4 GB
 - Hard disk: at least 85GB
- Software requirements
 - Window 10 64bit and above
 - Python IDE

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 Microsoft Word Document (.doc, .docx), and Text Document(.txt). The proposed system is tested with two file types as shown in the next section. The sizes of the files which are involved are varied between 1 KB and 635 KB and 80 numbers of files involved in these testing.

4.2 System GUIs

The implemented system includes two main parts. They are data deduplication and reconstruction. The graphical user interface for the implemented system's main window is shown in the following Figure 4.1.

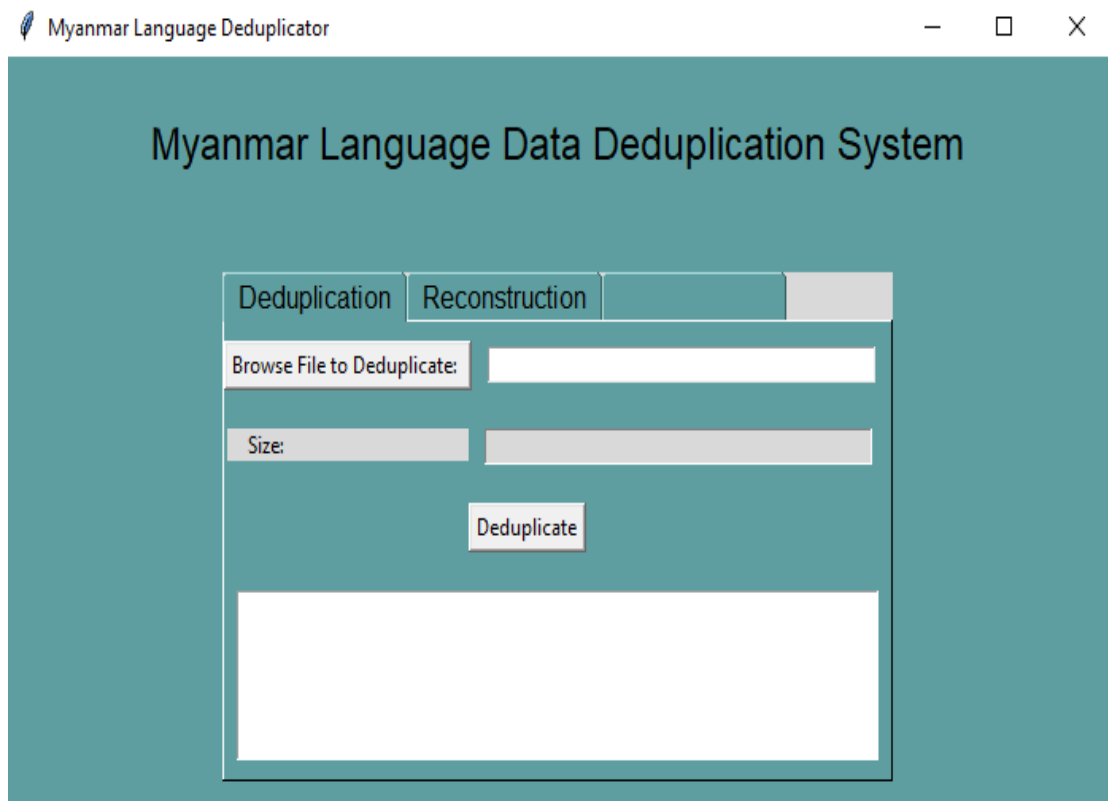


Figure 4.1. Main page of Myanmar language Data Deduplication System

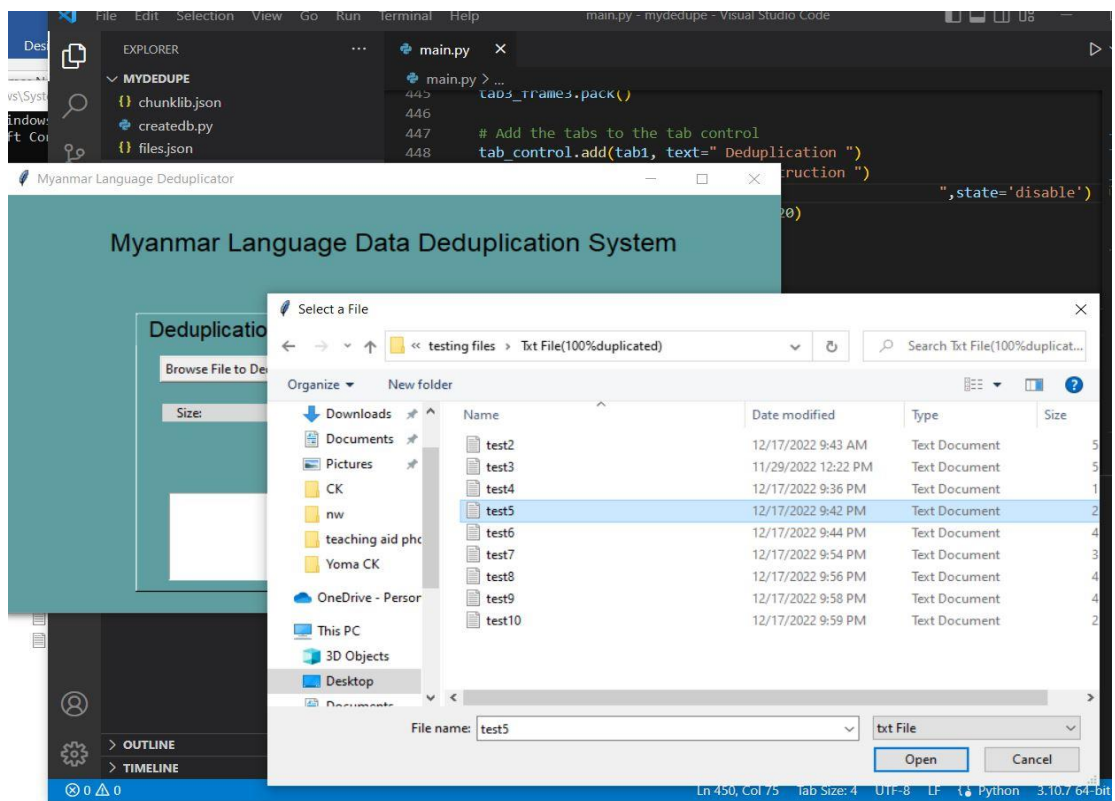


Figure 4.2 File selections to dedupe

In the above figures, if the user wants to store the text file contained Myanmar language have to click the 'Browse File to Deduplicate' button, then select the desired file from the file explore dialog box as in Figure 4.2. It will show the original file size of the selected file. Then, click 'Deduplicate' button to process. The output result will be shown as the following Figure 4.3.

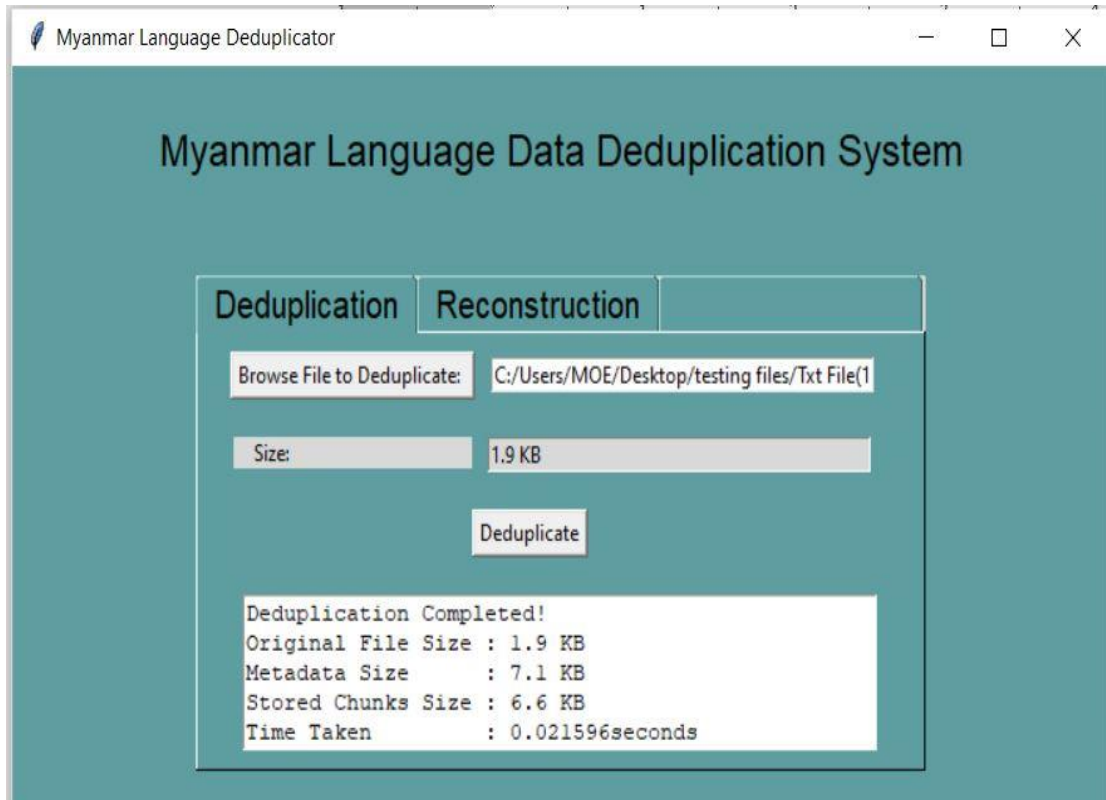


Figure 4.3 Deduplicated file result, first time store process

In the first time store process, if the deduplication is successful, the size of the selected file and the size of the metadata and chunk, which increased when the first deduplication was done, will be found in the text box. Then, the amount of deduplicated time is also shown.

In the second time store process, when saving the same file that has already been deduplicated, the metadata and chunk size do not increase. Also, the file size remains the same and the deduplication time is also faster. The output result will be shown as the following Figure 4.4.

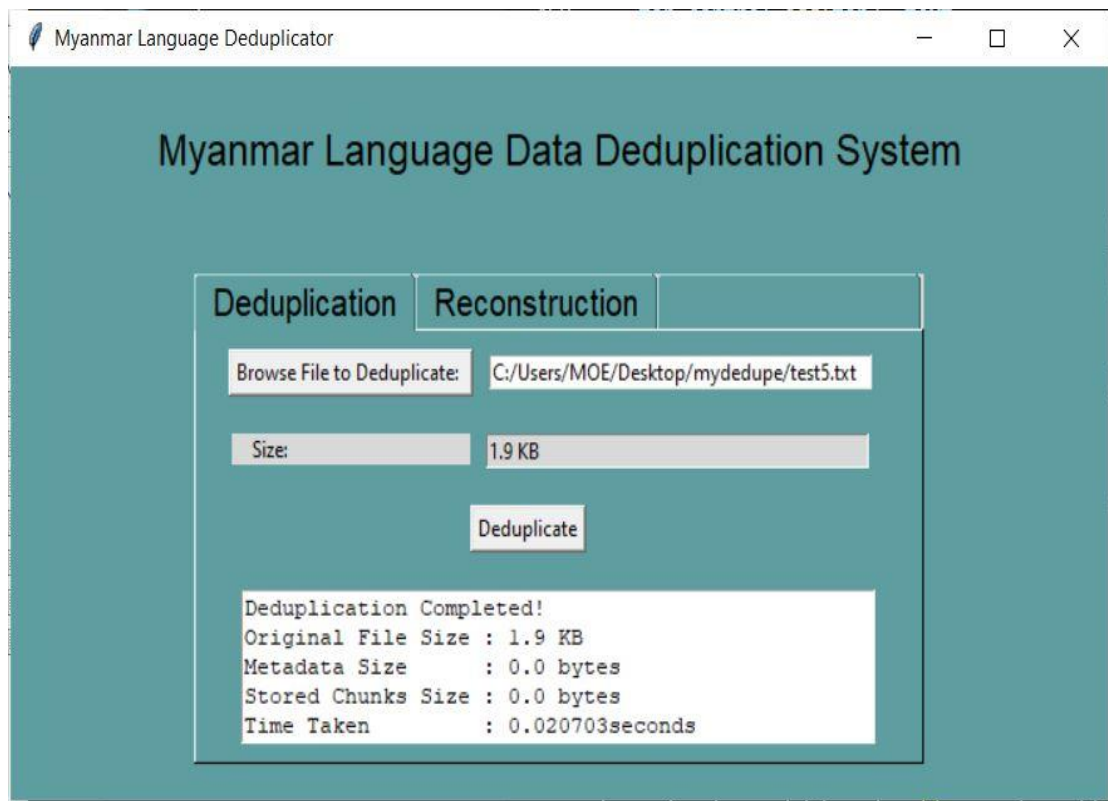


Figure 4.4 Deduplicated file result, second time store process

After storing the desired files to the data deduplication system, if the user wants to retrieve the stored file from the system, it needs to use 'Reconstruction' part. In reconstruction, the user can choose the desired file to reconstruct from the deduplicated file list as shown in Figure 4.5 at reconstruction phase.

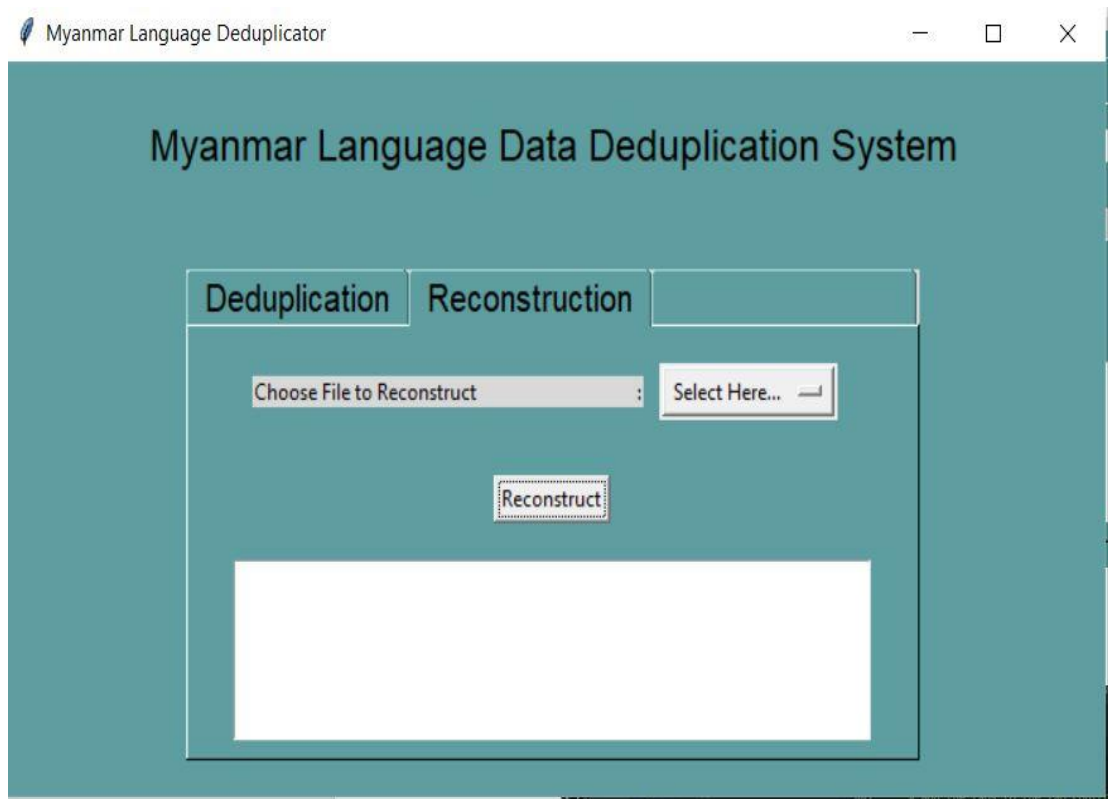


Figure 4.5 File Reconstruction

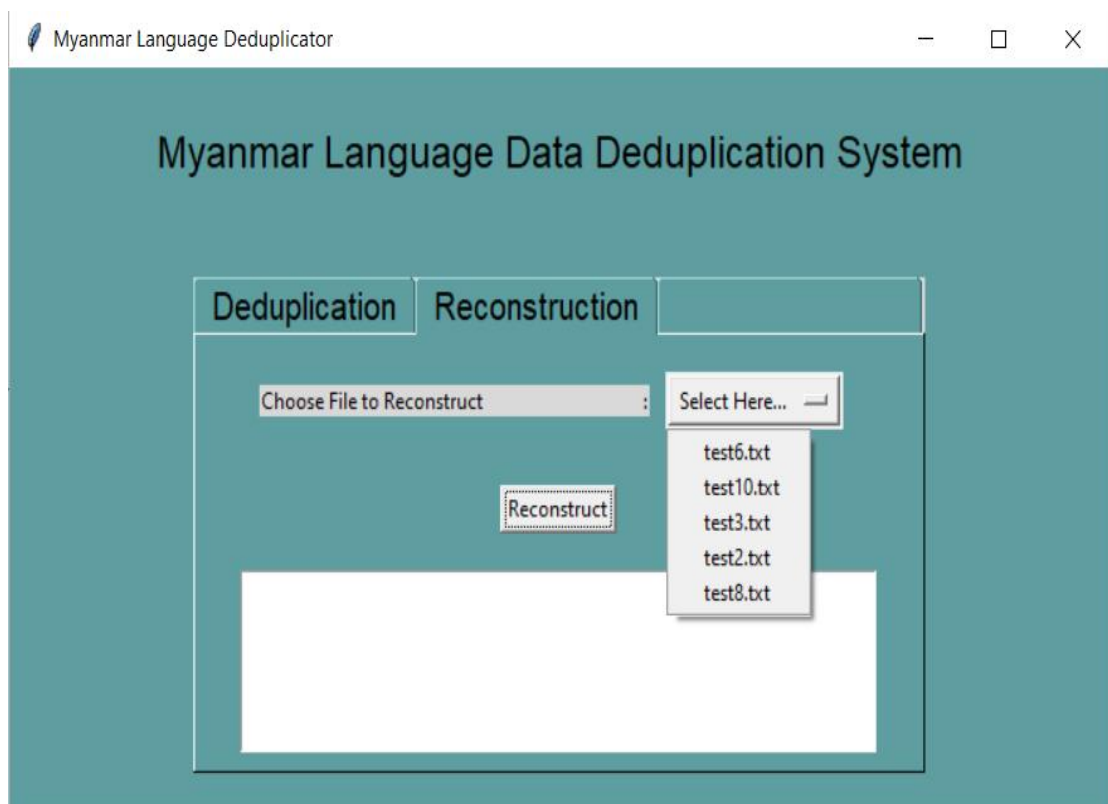


Figure 4.6 File selections to reconstruct

The above figure has shown that you can select the file you want to reconstruct from among the already deduplicated files. Then, click 'Reconstruct' button to process. The output result will be shown as the following Figure 4.7.

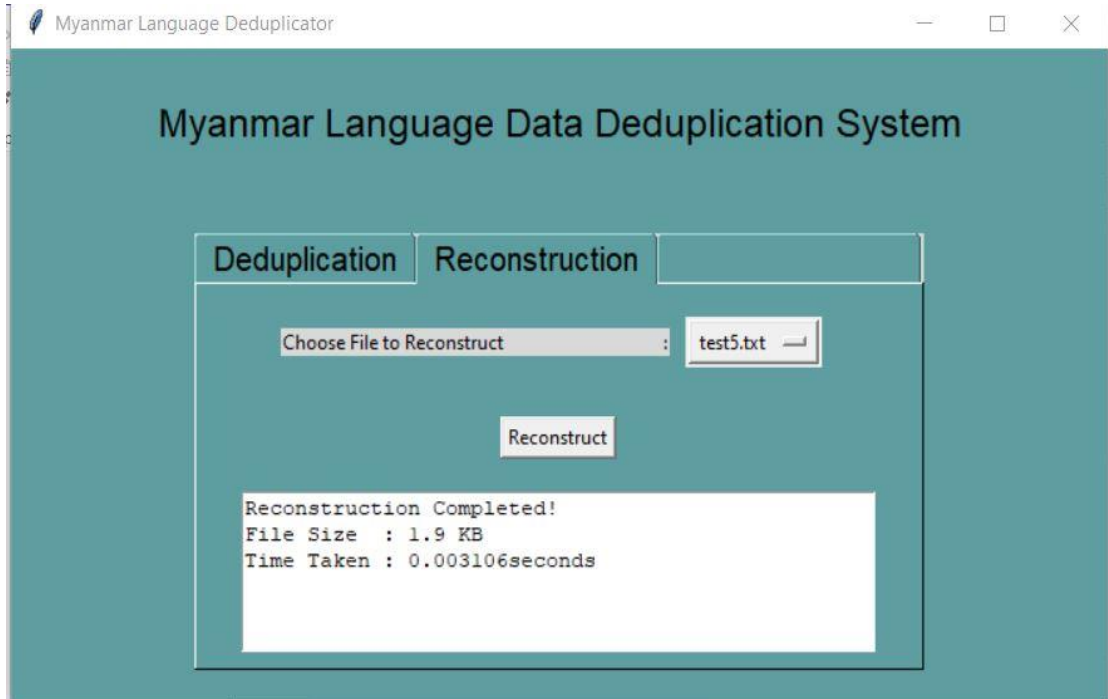


Figure 4.7 File reconstructed result

The above Figure 4.6 shows the reconstructed file. It is found out that the file size remained the same and the reconstruction time was faster when deduplication.

4.3 Performance Results and Discussions

This section illustrates the experimental result for the proposed system. It represents the performance in comparison bar charts for 1st time storage, 2nd time storage and 3rd time storage for the same content and also depicts the space reduction percentage according the evaluation method that is mentioned in section 2.10.

For 100% duplicate file (.docx, .txt): Tested results for the 100% deduplicate .docx and .txt files are shown in Figure 4.8 and Figure 4.9 below. File1 contains text content that has never been saved, and File2 and File3 contains the same content as File1. After duplicating the File1 for the first time, the storage of the ChunkLib increased and the storage of metadata increased. This means that no matter how many times you duplicate them, the storage size of ChunkLib and its metadata will not

increase. The size of files 1, 2 and 3 is the same as the contents of File1, so it is not necessary to worry about increasing file sizes.

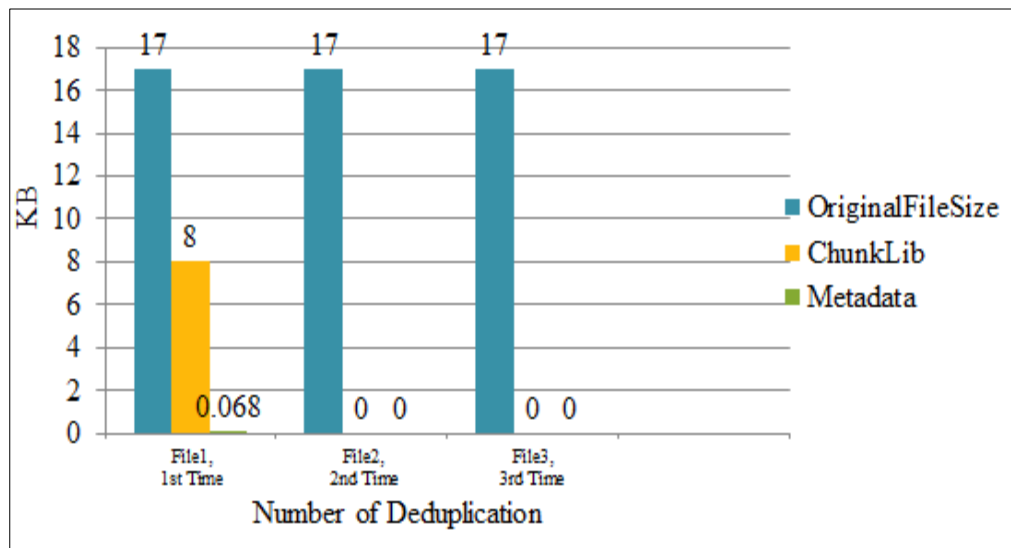


Figure 4.8 Deduplication for 100% duplicated file (.docx)

The space reduction ratio for 100% duplicate files is 50% for the first time deduplication. The calculation formula for the ratio is mentioned in section 2.10.

Bytes In= 17 KB (Original File size)

Bytes Out= 8 KB (Stored File size)

Space Reduction Ratio= $17/8 = 2.12$

Space Reduction % = $[1 - (1/2.12)] * 100 = 53\%$

For the second time deduplication and above will get fully deduplication and the storage space reduction ratio will become 100%. This means that the system can save storage space almost 100%. Even for metadata, it will only utilize very few bytes.

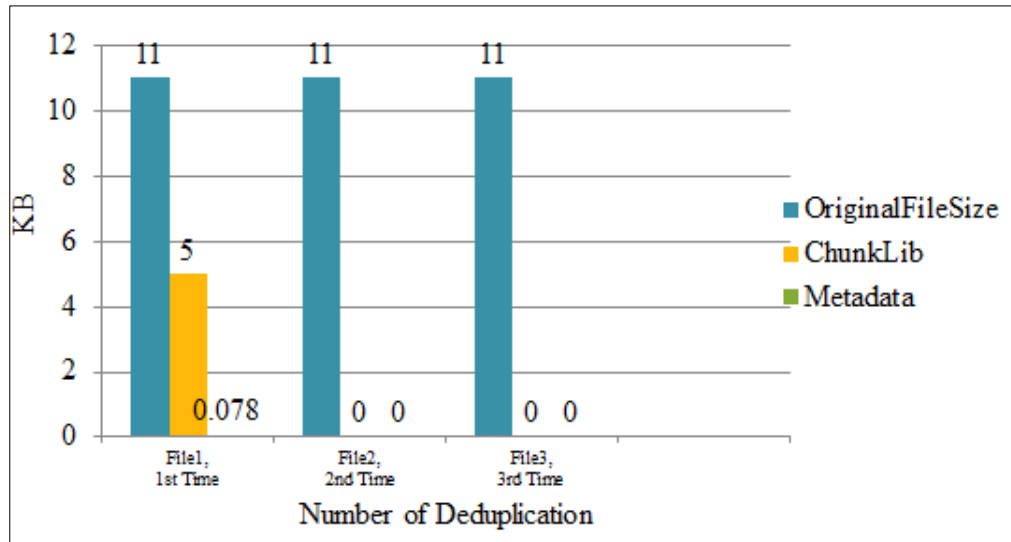


Figure 4.9 Deduplication for 100% duplicated file (.txt)

The space reduction ratio for the above deduplication process is calculated as the following.

Bytes In= 11 KB (Original File size)

Bytes Out= 5 KB (Stored File size)

Space Reduction Ratio= $11/5 = 2.2$

Space Reduction % = $[1 - (1/2.2)] * 100 = 55\%$

For the second time deduplication and above will get fully deduplication and the storage space reduction ratio will become 100%.

For 50% duplicate file (.docx, .txt): Tested results for the 50% deduplicate .docx and .txt files are shown in Figure 4.10 and Figure 4.11 below.

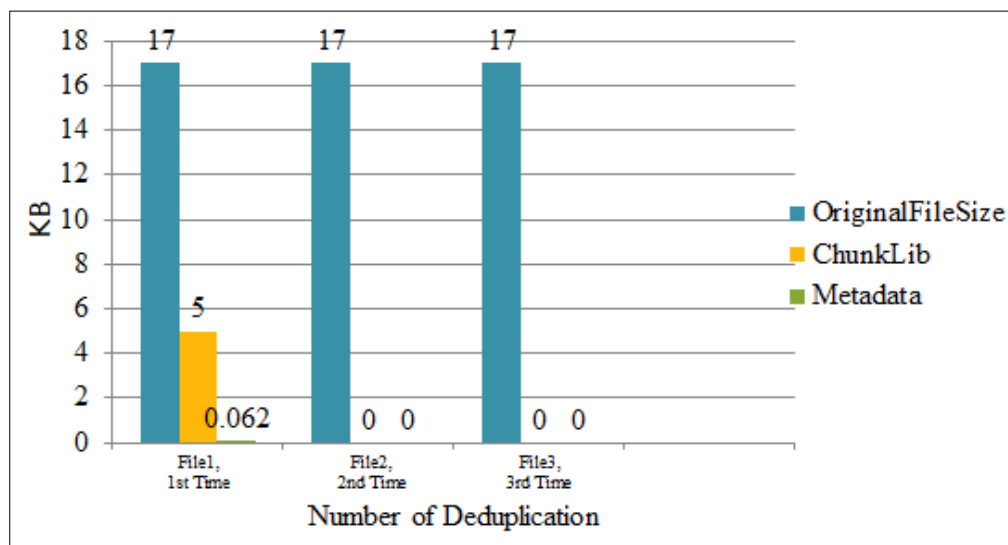


Figure 4.10 Deduplication for 50% duplicated file (.docx)

The space reduction ratio for the 50% deduplication is calculated as follow:

Bytes In= 17 KB (Original File size)

Bytes Out= 5 KB (Stored File size)

Space Reduction Ratio=17/5 = 3.4

Space Reduction % = $[1 - (1/3.4)] \times 100 = 71\%$

And the space reduction ratio for 50% redundant .txt file deduplication experiment is as mentioned below:

Bytes In= 11 KB (Original File size)

Bytes Out= 3 KB (Stored File size)

Space Reduction Ratio=11/3 = 3.67

Space Reduction % = $[1 - (1/3.67)] \times 100 = 73\%$

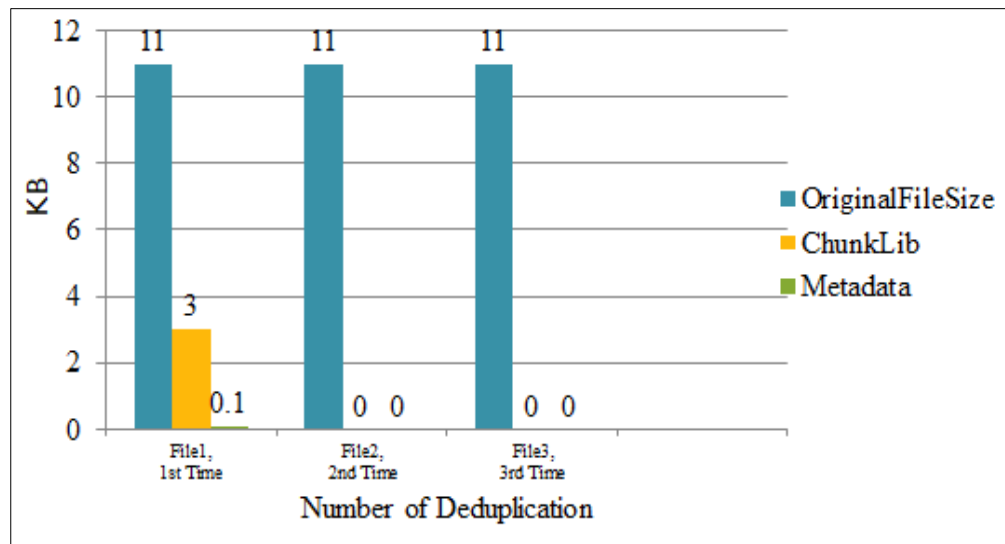


Figure 4.11 Deduplication for 50% duplicated file (.txt)

The size of ChunkLib and its metadata is only slightly increased when compared to a file tested in the lab. Figures show that the file tested here is half the size and contains the same content as previously saved and duplicated files, so the weight savings are significant.

For 25% duplicate file (.docx, .txt): Tested results for the 25% deduplicate .docx and .txt files are shown in figure below.

Since the content of this file is a quarter of the previously duplicated file, it can be seen that the chunks in the file have been decreased in size to one-fourth of

their original size in this image. Less than half of the total has been increased by ChunkLib.

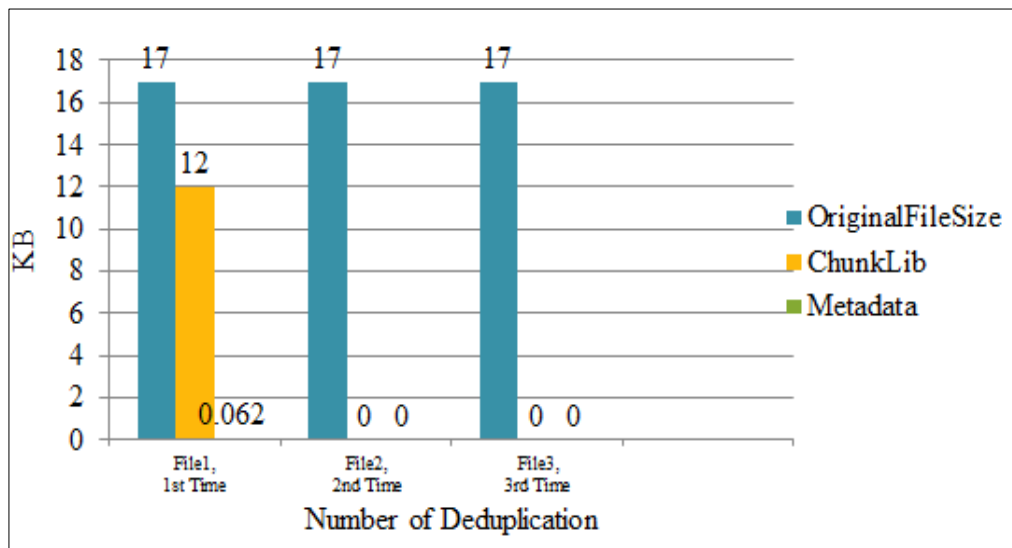


Figure 4.12 Deduplication for 25% duplicated file (.docx)

The space reduction ratio for the above deduplication process is calculated as the following.

Bytes In= 17 KB (Original File size)

Bytes Out= 12 KB (Stored File size)

Space Reduction Ratio= $17/12 = 1.42$

Space Reduction % = $[1 - (1/1.42)] * 100 = 30\%$

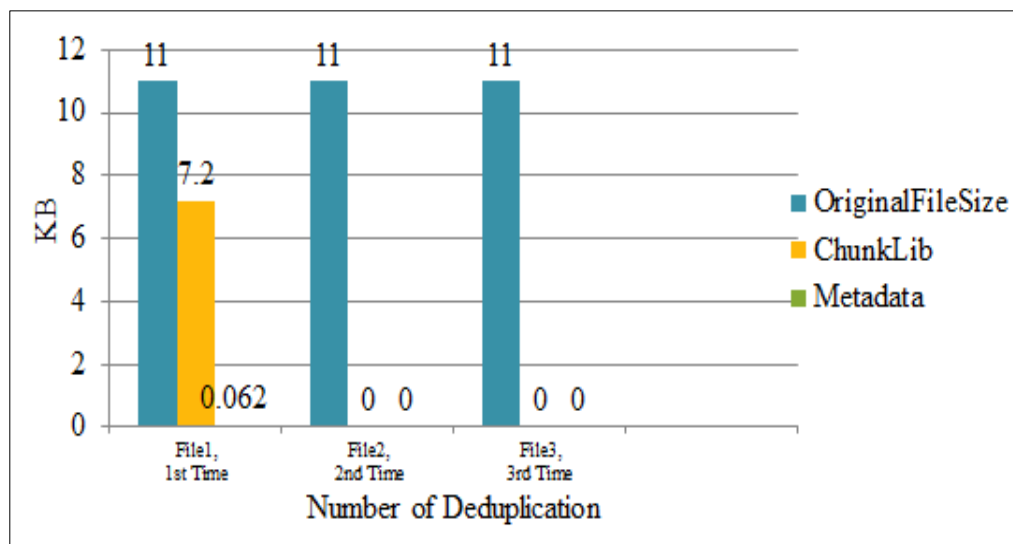


Figure 4.13 Deduplication for 25% duplicated file (.txt)

The space reduction ratio for 25% redundant .txt file deduplication experiment is as mentioned below:

Bytes In= 11 KB (Original File size)

Bytes Out= 7.2 KB (Stored File size)

Space Reduction Ratio= $11/7.2 = 1.52$

Space Reduction % = $[1 - (1/1.52)] * 100 = 35\%$

Time comparison for deduplication and reconstruction are shown in the following figure 4.14 and figure 4.15.

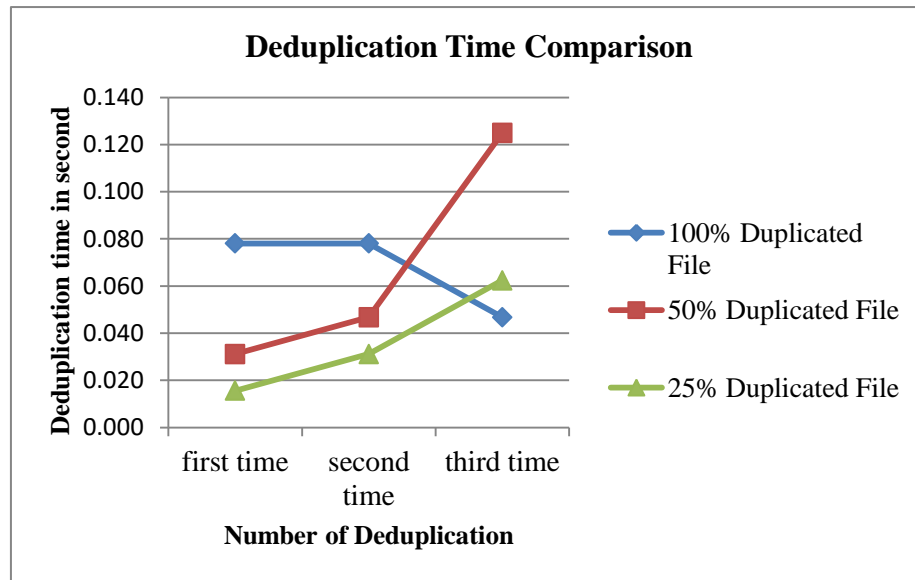


Figure 4.14 Deduplication Time Comparisons

Deduplication time for the first time store of 100% duplicated file is seen to be decreased for the second and third time store. But deduplication time for files that are 50% and 25% duplicates increases significantly with subsequent deduplication compared to the first.

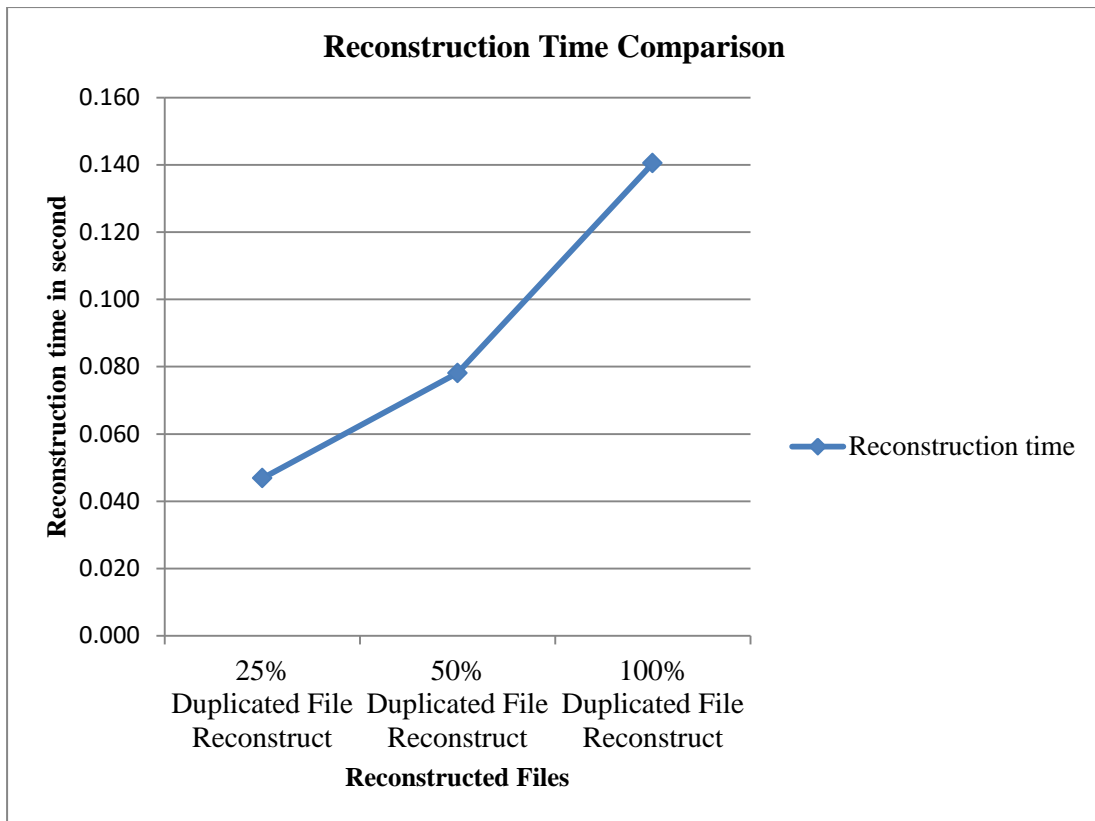


Figure 4.15 Reconstruction Time Comparisons

According to the above figure, it is found that the time to reconstruct a file with 100% deduplicated data is longer than the time to reconstruct a file for 50% and 25% deduplicated data. According to the findings, the more deduplicated data, the longer the reconstruct time.

4.4 Experimental Findings

If the user saves the same files for the first time, ChunkLib and metadata increase. If the files containing only one quarter duplicated data, $\frac{3}{4}$ storage space will increase. Moreover, if the user saves files that are half the same data as the previously stored file, the storage space will increase in only half of that file with metadata. Then, if the user tries to store the same contents as the previously stored file, the storage space will not increase, but only for metadata.

4.5 Summary

The Secure Hash Algorithm is used the experimental data deduplication of this chapter for Myanmar language storage, which measures the system's overall degree of completeness. The chosen test deduplication procedure is first demonstrated in this chapter for two different file types. These illustrate how chunk size and metadata

affect the deduplication time. The results of a few test deduplication experiments show that a deduplication system based on the secure hash algorithm (SHA1) and the duplicate finder algorithm can remove redundant data from a variety of file types while also using less storage.

CHAPTER 5

CONCLUSION

In this thesis, the proposed system can be used effectively by using data deduplication to store files written in Burmese language contents. It can reduce the storage space as its potential purpose. Eliminating redundant data can significantly shrink storage requirements and improve bandwidth efficiency. Due to the advantages of proposed ChunkID based Segmentation mechanism in this work, even the reconstructed file is deleted by the user; it can restore that file whenever it is needed as it retains the concerned metadata in permanent matter. Because of using effective Hash Algorithm for creating Chunk_ID to check duplicate data, it can be more effective in finding duplicate data in the existing storage.

5.1 Advantages of the System

- It can remove redundant data so that the storage space is reduced depending on how much duplicated data involved.
- It can obtain the efficient mechanism for searching identical data in the existing storage for Myanmar language by using ChunkID in terms of cryptographic hash function.
- The most interesting advantage is the recovery feature. Since the split contents are stored with ChunkID to the data deduplication system, even the original was deleted after storing in the system, it can reconstruct from the proposed system.
- Once the storing is done for the specific file, whenever the user want to get that file, it can be reconstructed from this system even after using the reconstructed file and delete again and again as it is permanently stored in the data deduplication system.

5.2 Limitation of the System

As the proposed system is based on content level data deduplication, it cannot work on other file types such as .pdf, .jpeg, .mp4 and so on. Such kind of data deduplication can only be done by using byte-level data deduplication.

5.3 Further Extension

In the proposed system, the duplicate checker is implemented with SHA1 to find out the identical contents in the text file, it can improve to Myanmar Language Plagiarism checker system with using additional technology such as centralized database technology which all academic publication papers are stored.

AUTHOR'S PUBLICATIONS

- [1] Thae Nu Aye, Tin Thein Thwel, “*Data Deduplication for Myanmar Language Storage by Using Secure Hash Algorithm*”, National Journal of Parallel & Soft Computing, Yangon, Myanmar, January, 2023.

REFERENCES

- [1] A.H.F. Laender, Altigran Soares da Silva,” Learning to deduplicate”, In Proceedings of 8th ACM/IEEE-CS Joint Conference on Digital libraries (JCDL), pp. 41-50, Association for Computing Machinery (ACM), New York, USA, 2006.
- [2] A Technical White Paper: “Data DeDuplication Background”.
- [3] "Collisions for 72-step and 73-step SHA-1: Improvements in the Method of Characteristics".
- [4] "Cryptanalysis of MD5 & SHA-1" (PDF).
- [5] "Crypto++ 5.6.0 Benchmarks". Retrieved 2013-06-13.
- [6] C. Liu et. al, “ADMAD: Application-Driven Metadata Aware De-duplication Archival Storage System”, In Proceedings of Fifth IEEE International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI), pp.29- 35, IEEE Computer Society, Los Alamitos, CA, USA, 2008.
- [7] Dirk Meister, “Advanced Data Deduplication Techniques and their application”, Dissertation Johannes Gutenberg University Mainz, March 2013.
- [8] Hla Hla Htay, Kavi Narayana Murthy (2008), “Myanmar Word Segmentation Using Syllable Level Longest Matching”, the 6th Workshop on Asian Language Resources 2008, pp. 41- 48.
- [9] K. Eshghi et. al., “High Performance Scalable Data Deduplication”, Storage Systems Research Center: University of California, 2008.
- [10] Matthew Brise, Quantum Gideon Senderow, “Data Deduplication methods for Achieving Data Efficiency”, Journal of SNIA Education Committee.
- [11] May Thu Win Moet Moet Win Moh Moh Than, “Burmese Phrase Segmentation”, Conference on Human Language Technology for Development, Alexandria, Egypt, 2-5 May 2011.
- [12] M. Jaehong, Y. Daeyoung, W. Youjip, “MUCH: Multithreaded Content Based File Chunking for Many Core Architecture”, In Journal of Parallel and Distributed Computing, Elsevier, Jan, 2010.

- [13] Myanmar Education Research and Learning Portal.
- [14] M.Lillibridge et al, “Chunk-lookup disk bottleneck/full chunks indexing encountered in in-line deduplication”.
- [15] Myanmar NLP, Myanmar Unicode Reference Documents and Research Papers (2006). Available at: [http:// www.myanmars.net/unicode/doc/index.htm](http://www.myanmars.net/unicode/doc/index.htm) (accessed 1 January 2007).
- [16] Myintzu Phyo Aung, Aung Lwin Moe, “New Phrase Chunking Algorithm for Myanmar Natural Language Processing”.
- [17] Qinlu He, Zhanhuai Li, Xiao Zhang, “Data Deduplication Techniques”, Department of Computer Science North-western Poly technical University Xi'an, P.R.
- [18] Raaed K. Ibrahim et al, “Implementation Of Secure Hash Algorithm SHA-1 By Lab view”, International Journal of Computer Science and Mobile Computing, Vol.4 Issue.3, March- 2015, pg. 61-67.
- [19] SHA-1 hash function under pressure – heist Security.
- [20] Tun Thura Thet, Jin-Cheon Na and Wunna Ko Ko, “Word segmentation for the Myanmar language”, Journal of Information Science, 34 (5) 2008, pp. 688–704.
- [21] Tom Sas – Hewlett-Packard, “Understanding Data Deduplication”, and SNIA Program in Storage Networking Industry Association, 2010.
- [22] Tin Thein Thwel, Ni Lar Thein, “Data Deduplication using B+ Tree Indexing”, Research in University of Computer Studies, Yangon, 2010.
- [23] Walter Santos, Thiago Teixeira, “A Scalable Parallel Deduplication Algorithm”..
- [24] Wikipedia. 2019. Burmese language Wikipedia page. [Online; accessed 18-July-2019].
- [25] Y. Tsuruoka and K. Tsujii (2005) “Chunk parsing revisited”, in Proceedings of the Ninth International Workshop on Parsing Technologies. Vancouver, Canada.

[26] Yadanar Oo, “JOINT WORD SEGMENTATION AND STEMMING FOR MYANMAR LANGUAGE”, Research in University of Computer Studies, Yangon, October, 2019.

[27] Z. Htut, Myanmar-Thai Co-workshop on Myanmar Language Implementation, Input Methods and Basic Encoding in Myanmar Language. Available at: <http://www.myanmar.net/unicode/doc> (accessed 1 January 2007).