Systemization and Evaluation for Data deduplication by deploying competitive chunking algorithm in polymorphic thread environment and avant-garde hashing techniques

MSc Research Project
Cloud Computing

Shilpi Madan
Student ID: x21145059

School of Computing
National College of Ireland

Supervisor: Rashid Mijumbi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Shilpi Madan
Student ID:	x21145059
Programme:	Cloud Computing
Year:	2022
Module:	MSc Research Project
Supervisor:	Rashid Mijumbi
Submission Due Date:	15/12/2022
Project Title:	Systemization and Evaluation for Data deduplication by deploying competitive chunking algorithm in polymorphic thread environment and avant-garde hashing techniques
Word Count:	6210
Page Count:	19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Shilpi Madan
Date:	14th December 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Systemization and Evaluation for Data deduplication by deploying competitive chunking algorithm in polymorphic thread environment and avant-garde hashing techniques

Shilpi Madan
x21145059

Abstract

From the past 3 decades the entire world is rapidly transitioning from the traditional analogue methodologies by adapting to more digital technologies thus marking the "Great Digital Revolution". Vast amount of data is created in several forms from digital footprints of consumers to development and research of new technologies across the globe. It is estimated that by 2025 almost 400+ exabytes of data will be generated globally every day. However efficient storage, management, and processing of this size of data can be a strenuous task for even larger corporations. There are several roadblocks towards attaining the efficient storage and utilization of data. "Data Deduplication" is one of the most efficient methods implemented to improve the storage abilities. This technique helps in identification and eradication of duplicate data. Due to the ability of identification of large levels of redundancy, 'CDC' better known as content-defined chunking is the key aspect of "Data Deduplication Systems". In this research paper we are focusing on optimization of "Data Deduplication systems, by analysis and updating of the current CDC parameters which will further enable efficient identification of chunk cut-points and fingerprint the dataset by applying a Novel- Hash function. In this research paper will also introduce the multi-threading content-defined chunking algorithm to enhance the computational process using multiprocessor technique. The given algorithm works on the concept of shifting window that slides one byte at a time in case there is no match with the hash value pool. Verifying the same on the AWS cloud infrastructure using different datasets and evaluating the average time for processing a file using parallel environment versus serial method. According on the findings of the research, our technique reduces execution time and the storage efficiency is increased by 70%.

Table Of Contents

1 Introduction

With the evolution of business in recent years, almost all the industries of our modern Society from Banking, Financial, Healthcare, to even traditional industries like Manufacturing adopted data driven business life-cycle processes. This rise of demand in data also created a scenario best described as "Data Flood", which is creating a big challenge for efficient Data storage and management protocols that were traditionally followed. Further with the sharp increase and growth of Cloud Services, "Data Flood" can have a severe impact on critical arrears of Data management like Data Transfer, Backup maintenance and Storage management. The process of "Data Deduplication" methodologies have proven to be a very effective solution to the problems arising due to this. It helps in identification of duplicate data or data with similar contents thus helping to control the data transfer costs in cloud environments, reduction in unnecessary storage and optimization of the network performance. "Data Deduplication" works at the very block level of a file and follows several steps of Chucking, Hashing, Fingerprinting and Managing the data based on the deployed algorithm of "Data Duplication". It breaks down the incoming stream of data into many data "chunks" that are each individually identifiable (e.g., SHA1 and SHA256). Through a process known as "content-defined chunking," the size of these chunks is determined and can either be fixed or changeable based on the content (CDC).As per the current search, variable-size CDC techniques are more successful than fixed-size ones at identifying redundant data and even avoid the boundary shift problem. The pictorial form of de-duplication is shown in below Figure 1

The basic aim of this research is to offer a chunking technique that addresses and solves efficiency and output challenges of previously used algorithms using multi-core processors. DER or 'Duplication Estimation ratio' is one of the crucial metrics used for

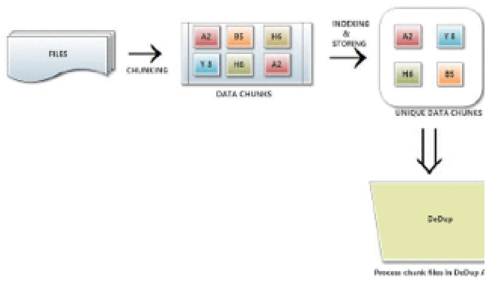


Figure 1: Data Chunks Formation

evaluating the result as it displays the percentage of data eliminated. To resolve the challenges arising due to them, we would propose to deploy Chunking and increasing the window size by shifting it gradually, to observe the efficiency outcome of Data De-duplication methodology in a cloud environment. Its primary objective is to improve the average chunking time what we can achieve through parallel architecture. This will have low computational overhead and will have a higher deduplication throughput than the previous one without affecting the duplication ratio. The key focus of our proposed work will be retaining both the DER and the primary of the input data stream. Performance is also determined by chunking throughput, which is simply the count of bytes every second. You may determine the amount of duplicates by dividing the total number of handled data files by the result multiplied by the throughput. Data-deduplication is an expensive task specially Content-defined Chunking and calculating Hashing index stage requires more processing time and are CPU intensive, that eventually leads to bottleneck in its performance. To work on this issue here in this paper we are proposing parallelism in de-duplication technique that will focus to parallelize the chunking stage, where entire dataset will be divided into various segments and further CDC will execute on individual segments parallelly on seperate thread and afterwards re-chunk & enrols the boundaries of the corresponding point. This is based on MapReduce functionality, which ensures that output of parallel CDC is similar as sequential one. Incorporating cloud services for any business judgment necessitates controlled resources, and cloud storage is no exception, as backups kept in the cloud and later utilized for restoration must ideally be contradictory free. The copy or clone may be uploaded into the system after restoration, guaranteeing that the data is not destroyed and may be utilized and kept safely. As a result, we will implement a data de-duplication mechanism in our cloud platform using several AWS services.

1.1 Motivation and Research Problem

Goal of this research article is to propose a superior Content Defined chunking approach than the prior one, as well as a suitable Hashing algorithm employing the notion of multi-threads. This even discusses how, with the aid of threads, they may improve total chunking efficiency and its influence on average chunk time. The use of this method will boost cloud storage management since no repetitive data will be transmitted over the cloud, hence saving network capacity. This investigation gives an answer to the following question:
Data de-duplication Systemization and evaluation using a competitive chunking algorithm in a polymorphic thread context and cutting-edge hashing techniques

1.2 Report Structure

Section 2 presents a thorough evaluation of numerous Chunking algorithms, including their limitations and strengths. It also provides all of the features required to transmit data to AWS data storage and perform de-duplication on the provided content, such as security and encryption procedures. Section 3 outlines the mechanism for de-duplication, followed by the appropriate chunking technique and hashing algorithm with multi-processors. The planned study and its execution on the AWS cloud are covered in Section 4. Section 5 describes the assessments of supplied datasets based on two factors.

2 Related Work

Initially Huffman coding and LZ Ziv and Lempel (1978)compression were the most common ways to manage redundant data. Due to its time-consuming nature and as it is specific to certain amount of data, de-duplication came into picture. De-duplication is a technique of deleting redundant information using Hash value and deriving chunks with the help of various chunking algorithm. Logical pointers are being used to maintain the duplicate data and the chunking is further classified into fixed and variable sized chunks. Content Defined chunking was introduced to get rid of drawback defined by fixed sized chunking of boundary shift that arises in the event of a byte expansion or removal by Yuan and Yu (2013)Yuan and Yu (2013) to obtain cut-points for further chunking, where Venish and Siva Sankar (2016)Venish and Siva Sankar (2016) assessed several chunking algorithms that focus on its performance and efficiency in case of chunked material of exact type.

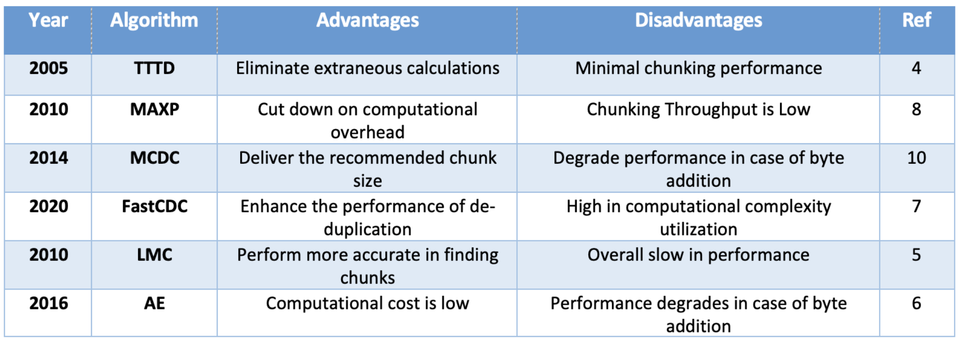
2.1 De-duplication Chunking Algorithm

Eshghi and Tang (2005)Eshghi and Tang (2005) devised TTTD (Two Threshold and Two Divisors) to solve previous shortcomings. It has dual divisors, one as the primary divisor and one as a standby, which aids in obtaining the stoppage if the primary divisor misses to do so. The limitation of TTTD is that the primary divisor value isn't really content-based, but rather an expected value, and the standby denominator is not used until it exceeds the maximum cutoff point, which is usually close to Tmax, which creates unnecessary computation and tends to make the methodology computationally expensive.
Local Maximum Chunking (LMC) by Bjrner et al.(2010)Bjørner et al. (2010a), which focuses on discovering the local maximum valued byte by employing the same dynamic panel model, will seek the local maximum value somewhere around two fixed size windows. The cut-point is defined whenever the derived identity is larger than the number of bytes in that adapting. The drawback is it matches the byte range every-time the window shift takes place and that result to be slow.
Zhang et al. (2016)Zhang et al. (2016) introduced Asymmetric Extremum(AE), to overcome LMC which operates on bytes and treats them as digits. It picks 2 windows, the left side is variable-sized and the right is fixed-sized, excessive value is found in between window and the cut-points too. It increments the window size. It has low computing needs, but is it not appropriate for byte shifting.
FastCDC Xia et al. (2020) chunking is the most recent CDC approach presented by Xia et

al.(2020), that follow gear driven rolling to derive hash value along with hash judgement and primarily it ignores cut-points and works on minimum chunk size. This algorithm rolls 2 bytes at a time and separates the even and odd bytes that makes the process faster by 1.5 to 3 times, however this leads to low duplication ratio.
Nikolaj et al. (2010)Bjørner et al. (2010b) developed the MAXP method that computes the hash value before-hand, it takes bytes directly as digits and to locate local maximum values it scans entire fixed-size symmetric windows which eventually becomes chunks. The limitation occurs when any insertion takes place this algorithm scan the window back and forth leading to low throughput.
Kruus et al. (2010) created the Bimodal MethodKruus et al. (2010), which is an modernized version of the original CDC algorithm. Entire datastream is divided into larger chunks and turn into smaller pieces, in case of undetected redundant, when the information that is now altered is present in both chunks in that location, its size will be updated, and that comparably little unobserved information within this altered region will be thrown and developed into a new variety.
MCDC seems to be an upgraded variant of Bimodal described by Wei et al. (2014)Wei et al. (2014) its dependency is on (CR), it determines the unchanging size of information, and the ratio is categorized into three separate variants, and the three chunk sizes are set based on these ranges. The fundamental problem of this method is that it is not resistant to boundary shift because the portion size is set only by three rectified limits. Chunk size is derived using Uni-modal technique. With the chunk size the CR is defined, because of increased workload, the efficiency of this method is poor, with a very low throughput value.
Yu et al. (2015) presented Leap Based chunkingYu et al. (2015), it is implemented on randomly transformed to confirm if the defined slide is competent or not, and it eradicates the functionality of rolling hash. This chunking technique analyzes the window overall satisfaction using two parameters, size and power and from them we can determine which CDC performs best.
Kaur et al. (2018) Kaur et al. (2018) grasp all of the numerous data de-duplication strategies in use and categorize them based on the method they have adopted for cloud storage. In addition, this research classed data de-duplication algorithms that deal with texts and other forms of multimedia data streams, outlining their issues and identifying their limits.
The Low Bandwidth File System (LBFS) (2001)Muthitacharoen et al. (2001) is another sort of CDC algorithm developed by Muthitacharoen et al. Because the main purpose is to decrease border displacement difficulties, it separates the input files flow into pieces based on its content. It additionally functions on the file data directly, assessing the checksum or, in all the other cases, fingerprints of the suitable frame using a template matching approach and splitting the chunks depending on its split, which are defined by satisfying certain predetermined specifications. Nie et al. (2019)Nie et al. (2019) developed an analysis in which the size of clustering block could be optimized in order to boost the effectiveness of dedupe techniques, whilst taking into account that the size of the clustering block ought not to be far too large or far too tiny. To increase the removal of redundant value, the block size used during the chunking operation must be optimal.
Ni (2019) Ni (2019)elaborates on the current CDC methods and developed a new two-phase method (SS-CDC) that has significantly improved chunking performance and obtained the very same repetition proportion as conventional and concurrent approaches. By employing weak signatures for detecting copying using fast, hash value that doesn't

collides with the existing values.
Gholami Taghizadeh et al (2020)Gholami Taghizadeh et al. (2020) study provided a new advanced method of flash memory de-duplication. The write requests were classified according to the type and substance of the data, and the related metadata was separated into categories to aid in the search process. This shortened the time consumption and considerably enhanced the rate of de-duplication.
Guo et al.Guo and Efstathopoulos (2011) notifies an incident-driven diversly client-server multiplexed interactivity model to increment the throughput of the fixed chunks certified de duplicative simulation system. It also sets forth mod us of border indexing specimens and classified mark and sweep to amend and enhance the attainment benchmark and extendibility of deduplication system. Ma et al.Ma et al. (2012) proposes an adaptive and robust channelized compute-accelerated subtasks of fingerprinting, transcoding etc in a rigid chunking based de-duplicative simulation system. Liu et al. Liu et al. (2009) imitate fingerprinting computation for de-duplicative simulation by first partitioning each data set into several chunks then fingerprinting synchronously each set and merge the fingerprints as the chunk.
Below is the circumscribed Table defining the observations made on distinct chunking methodologies based on its leverages and subservience's. In this suggested paper we will be evaluating the functions of the algorithms which best performs on variable sized chunks and will augment the window size by 1 byte at a time and configuring the same on diversely threaded environment to enhance the time efficiency of chunking process. The implementation would be capable of delivering an exquisite throughput while staying elusive from the effects of chunk volatility.

Table: Summary of reviewed Chunking Algorithms



Year	Algorithm	Advantages	Disadvantages	Ref
2005	**TTTD**	Eliminate extraneous calculations	Minimal chunking performance	4
2010	**MAXP**	Cut down on computational overhead	Chunking Throughput is Low	8
2014	**MCDC**	Deliver the recommended chunk size	Degrade performance in case of byte addition	10
2020	**FastCDC**	Enhance the performance of de-duplication	High in computational complexity utilization	7
2010	**LMC**	Perform more accurate in finding chunks	Overall slow in performance	5
2016	**AE**	Computational cost is low	Performance degrades in case of byte addition	6

3 Methodology

Data Deduplication is one of the foremost necessity for eradication of corresponding data to maintain optimum requisites for achieving an efficient storage services in IT architecture. Deduplication methodologies not only plays a paramount role to address the rising cost of cloud storage but also helps in cultivation and support for accelerated backup speeds and optimizes the data transmission rate by eliminating duplicate data across the computing grid structure. In this paper we would like to introduce an algorithm primarily positioned augmenting every byte in order to improve divergent onset and a cluttering function that operates on the very principle of multi layered cluttering behaviour. This will help us achieving the ability to reduce the probable hashing collisions without any adverse consequences on computing abilities of a network. In order to expedite the de-duplication algorithms, a diversely threaded de-duplication methodology is introduced in this paper where the CDC modus operation will be performed co extensively on manifold datastreams along with this on Hashing stage.

3.1 Architectural Diagram of De-duplication

The procedural manoeuvre for the architecture of deploying de-duplication methodologies must be followed throughout irrespective of IT environment cloud or on-premise. These standards processes involve multifarious steps involving chunking the manifold of datastream, estimation and computation of its hash value and ensuring the hash must correspond with the current Hash Database. Below is the outflow of the steps in form of a diagram Figure 2 with excerpting all the stages.

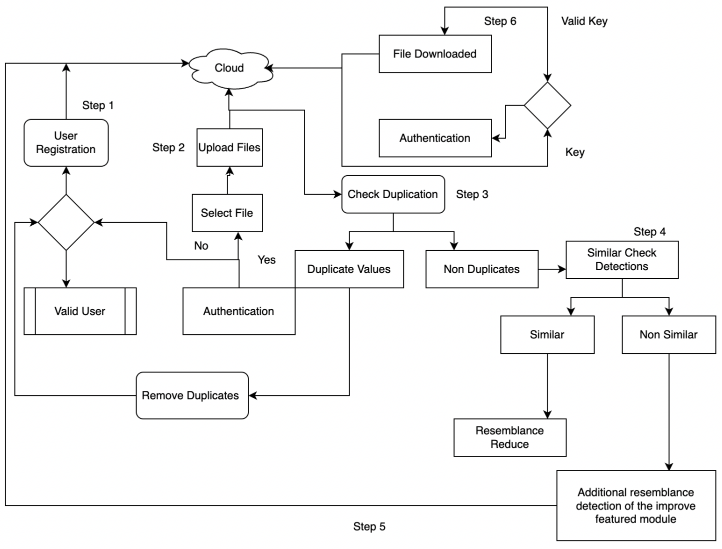


Figure 2: Data-Flow in De-duplication Architecture

Stage 1: Intendment of the data stream.

Stage 2: Chunks will be collected respective to the suggested chunking algorithm. The data stream will be bifurcated and branched into chunks. Further the hash value will be determined for using chuck by utilizing the in-use hashing algorithms.

Stage 3: Post-determining the chunk value, a chunk identifier is derived for every individual chunk that consists the size of the chunk, pair divisor value and unique 3 hash values.

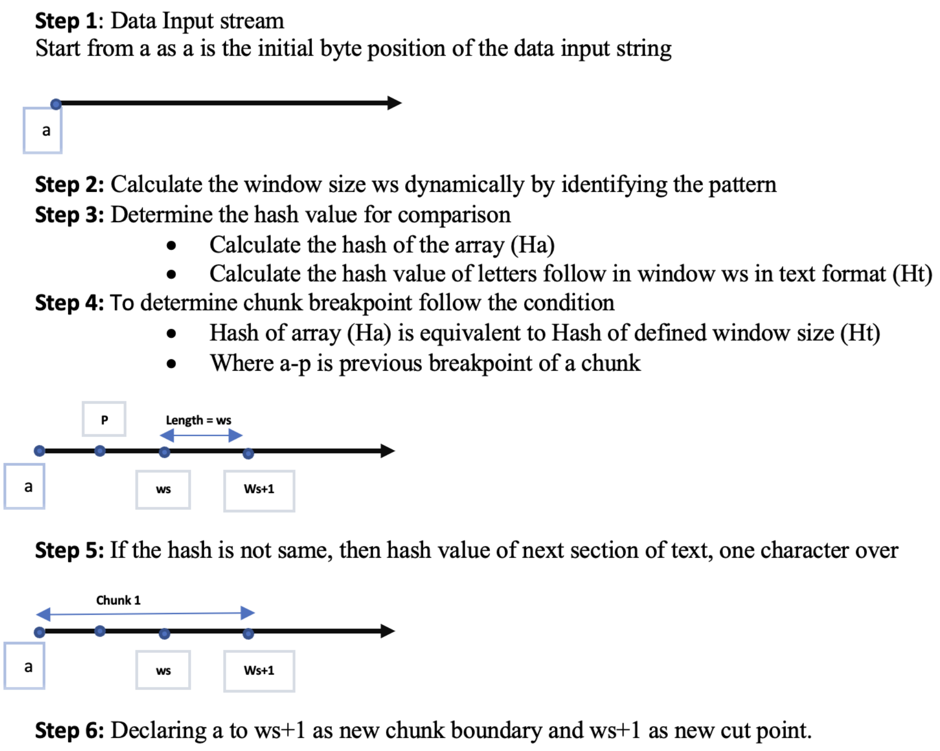
Stage 4: The determined hash value is analysed in the fingerprint repository to detect any similarity in the chunk. In case the hash value matches, the logical pointer points the originating chunk datastrem and stores the logical address.

Stage 5: In the scenario if no match is determined, it stores the chunk of the datastream in unique data storage container also further creating a reference in the hash table and add its augment its hash values generated in Stage 2.

3.2 Proposed Chunking Algorithm

The CDC represents the most truthful chunking strategy used for data de-duplication; it absolutely depends on the circumstance that is currently characterized in the algorithm and, predicated on just that affliction as well as the information referenced in the data-set, it determines the portions that are the information on which to base utilised for the further window interpretations. CDC has assisted in overcoming a significant key constraints of chunking, that was a component of predefined length or deterministic chunking technique called border tilting, which occurs when there is any addendum or withdrawal of one bit of data in the information stream of data, resulting in a new chunk and different value of cubes with distinct signatures or hash values, that also ultimately has become an innovative set of information. The below is the description and the step by step explanation of proposed algorithm with the assumption of data string of specific length, and calculation of hash value is derived using the hash function on the later stage of this section.

To calculate the fingerprint data bytes are entered into a feature space. If the derived



Step 1: Data Input stream
Start from a as a is the initial byte position of the data input string

Step 2: Calculate the window size ws dynamically by identifying the pattern
Step 3: Determine the hash value for comparison
- Calculate the hash of the array (Ha)
- Calculate the hash value of letters follow in window ws in text format (Ht)

Step 4: To determine chunk breakpoint follow the condition
- Hash of array (Ha) is equivalent to Hash of defined window size (Ht)
- Where a-p is previous breakpoint of a chunk

Step 5: If the hash is not same, then hash value of next section of text, one character over

Step 6: Declaring a to ws+1 as new chunk boundary and ws+1 as new cut point.

value satisfies the specified value, the cut-off is established as the position of the shifting window's final byte. Otherwise, move the window size by one byte ahead and repeat the hashing computation and fingerprint comparison. When all of the cut spots in the file have been discovered, the algorithm ends. Here, we will be initializing various parameters width of the window size, comparable hash value, further the defined window is divided into segments to create chunks. The algorithm derive the chunks within the input file and compare its hash value with the existing hashes to find the duplicate chunk. The previous value of fingerprint is taken as a base to calculate for the new window that has shifted by 1 byte.

$$(A_1, A_2....A_s) = (\sum_{a=1}^{s} A_a P^{s-a}) mod C \tag{1}$$

To calculate the new fingerprint using the previous value we operate on above function where C is the size of the chunks, s is the window size for the input stream marked from A1 to As. For the next byte we will take the values with adding the next integer in that string value.

3.3 Chunks Generation using Multi-Threads

Data-deduplication is an expensive task specially Content-defined Chunking and calculating Hashing index stage requires more processing time and are CPU intensive, that eventually leads to bottleneck in its performance. Mapreduce is one way to execute content chunking in multi-threads that computes on multiprocessor approach setup on multi-core system to work on numerous split but continuous sub-streams referred as “segments” extracted from same input dataset usually of length 1 and 2 MB. The first stage of parallelization is pipelining the steps of deduplication - Content Defined chunking(Stage1), Securing the hashing index(Stage 2), Scanning the stored hash values(Stage3), Storing Chunks and index values(Stage4). Once the Pipelining is done processors are assigned for the Chunking stage where it needs to divide the input-stream into fixed sized segments following the size protocol of predefined value of maximum size of the chunks and the process is called “Map”. Parallelly the chunking algorithm will execute in these data chunks and the chunks will derived along with its hash value. Another step is to combine these chunks correctly following the chunking cut-points rule this is called “Reduce” function. The bifurcation of input streams into multiple segments and further combining those segments after the operations is illustrated below Figure 3 The two segments are running

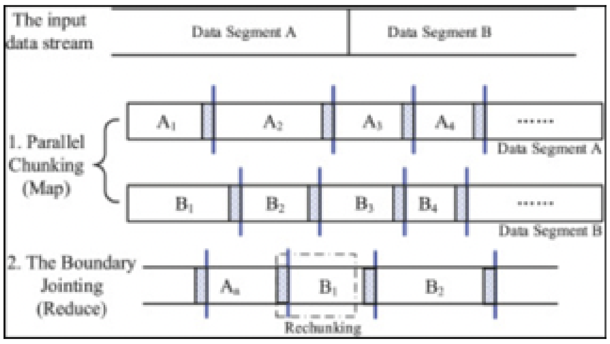


Figure 3: Illustration of MapReduce

CDC algorithm synchronously in order to generate chunks arguably the last chunk in the segment will be considered in preliminary piece as the final break will potentially meet the standard CDC protocol as displayed in the diagram. The valid “cut-point” for the chunk is inside and we will re-initiate the chunking for alteration as the 2nd step for completing parallelization.

The concurrent CDC operates simultaneously with dual processor.This process is mapreduce: the input file is diagrammed into 2 divisions and edges are re-chunked for correction.

3.4 Proposed Hashing Technique

Authentic Hashing technique used for deduplication process, consume more storage and is time consuming as it needs to eradicate hashing collision separately. This issue is very common in SHA1 and MD5 algorithms that calculates hash value of size 160 bits and 128 bits respectively. In this paper we are proposing layered hashing algorithm with our proposed chunking method, layered hashing function that relies on a hash value for the sequence representation is constructed using a polynomial limit total of a sequence of non-repeatable null bytes repeated by a random integer series. Having 3 hash values for a unique chunk clubbed with the respective fingerprint leads to have less chance of hash value collision results in chances of unique chunks to identify redundant value. This approach is less complex and more secure in nature as compared to previous used hashing methods.

Hash value derived through this method is very small, each hash is of 16 bits that makes a total of 48 bits for all 3 hashes generated for a unique chunk, that will eventually generates smaller Chunk ID which requires storage space in hashing table. The overall benefit of implementing this hashing to improve the throughput and spare some space used by hashing table.

Below is the Proposed Hashing Method and the function it is based upon: In this we

```
Hashing Algorithm

To obtain three distinct hash functions one per chunk.

Input:  Chunk as array of bytes (Ca)
        Chunk Length (Ck)
        Hash1, Hash2, Hash3 (HV1, HV2 & HV3): 3 arrays of 255 random values

Output: Hash values

Step1: Initialization H1 ← 3, H2 ← 37, H3 ← 17
        Li ← 0
Step2: Compute three hash values for the chunk
        For I = 0 to Ck-1 Do
        Li = Li +1
        If Li > 255 Li = Li - 255
        H1 = H1 + (HV1[Li] * Chunk[I]) If H1 > 0xFFFF H1 = H1 & 0xFFFF
        H2 = H2 + (HV2[Li] * Chunk[I]) If H2 > 0xFFFF H2 = H2 & 0xFFFF
        H3 = H3 + (HV3[Li] * Chunk[I]) If H3 > 0xFFFF H3 = H3 & 0xFFFF

End For
```

are taking chunk bytes, defining the length of a chunk, Hash Value (HV) is an random number generated using rand() function which will be saved and used repetitively. The usage of a distinct sequence of random integers to generate multiple short hash values is sufficient to generate unique signatures to describe chunk plain text contents. Furthermore, the computation used to calculate the hash values seem to be relaxing; just primitive calculations are used, which makes the required cost of computing the hash is reduced in as compared to traditional method additionally, the size of hash is in bits and bounded independent of big sized data.

4 Implementation

The implementation of this research methodology is performed to achieve efficient results in time taken to execute chunking on various datasets together with ensuring appropriate hashing methodology that truncates probable hashing collisions. Here, the algorithm is implemented in Python Version3 with multi-threading functionality, it is deployed on AWS cloud infrastructure to understand the efficiency of both serial and scalable chunking techniques on a same set of input stream.
To setup the experiment we have created Ubuntu 20.04 instance on AWS i386 that runs on 2.5GHz clock speed, code used to perform chunking is written in Python3.7 combined with layered hashing algorithm. We will be performing our evaluation using different numbers of processor and observer the average chunking time, along with the throughput, this is further generate hash values that is later used to compare the redundant data.

4.1 Content Defined Chunking

The primary function to perform any de-duplication is to identify the patterns that are divided into chunks, the goal is to construct a limited and predetermined data window on the input set. Once the window is identified, the 3hash values will be calculated using our hash function and compared with the existing hash table. Chunk patters Krishnaprasad and Narayamparambil (2013) is calculated with below general linear model:

$$Value(a, a+(W_s-1)) = (N^{W_s-1})*D_w[a]+(N^{W_s-2})*D_w[a+1]+....+ND_w[a+W_s]+D_w[a+W_s-1] \tag{2}$$

Here, array of data is $D_w[]$ and the given window size is W_s, we are considering starting point of the input data from a and calculates the value till a in addition to size of the window-1.
Below is the flow diagram Figure 4.1 of the overall execution of chunking process and the comparison of hash values.

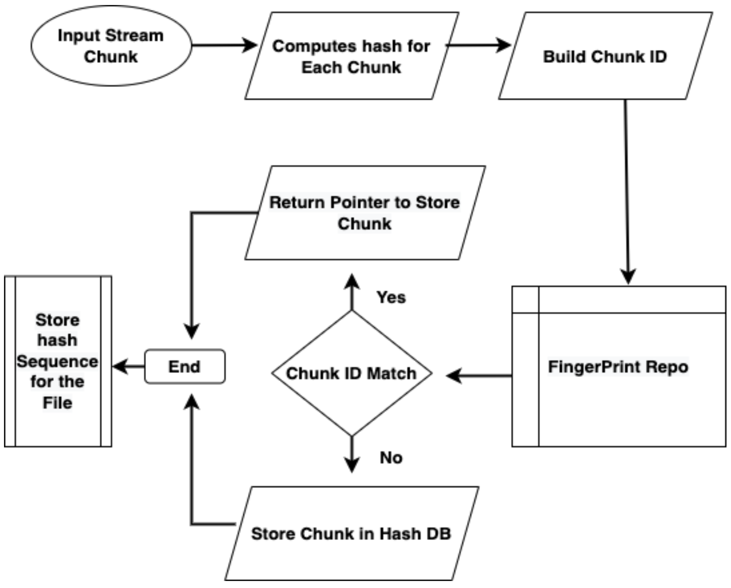


Figure 4: Flow of Chunking Process

The primary functionality of the chunking algorithm is to identify the smallest chunk in that input string and then generate the hash value for every chunks. To retrieve the precise hash value, we evaluate it to the predetermined fingerprint, and if they are the identical, we may declare it as the needed chunk boundary; if they are different, we rebuild the window by moving it from the value to right of the existing cut point. By shifting is right we need to re-calculate the hash for this new chunk point and this procedure continues, till we get the desired result that is when the hash value is equivalent to the saved fingerprint. To achieve chunks using above concept for any particular dataset, we have written below algorithm in Python.

Algorithm:

Input:
- dataset
- present_hash_value
- length #Window size

Output:
- Chunking Breakpoint: cut_point

Function

```
CDC (dataset, present_hash_value, length)

cut_point = 1
index=0

while (byte = Read_byte(dataset)) do
array[index%length+1] = byte
if (Size(array) >= length) then
if (Hash(array, index, length) == preset_hash_value) then
return cut_point
end if
else
cut_point = cut_point+1
end while
```

4.2 Implementation on Cloud

The architecture is divided into two parts: client and cloud. Local data de-duplication is performed by the client components. The fingerprint index of the data chunks is generated by the client computer. When a client uploads data to the cloud, compressed portions of the material are transferred to cloud storage. The fingerprints of the received index are saved in the Global index on the cloud side.

- The small files usually less than of 10KB are first identified by a filter and data de-duplication is directly performed on them whereas large files needs to pass though the chunking unit for further chunks division, that divides the files into chunks of 256KB each.
- These chunks are further delivered to the hashing unit that calculates individual hash values, our proposed hashing technique will generate 3 hash values for respective chunks.
- We will first match the generated hash value with the index table present in index generator locally to identify any redundant data, if there is any then it will not commit in Cloud storage. It there is no match between the existing fingerprint and

the new value, the chunk will further pass through the compression unit based on DEFLATE, that compressed chunk will be delivered to Cloud.

- Incoming file chunks are sent to the chunk index based on the file information. Each item in the index comprises the fingerprint value, chunk's length. The SQL index is used to store data chunk information and to find comparable data chunks based on their fingerprints. Because the SQL index can traverse the database faster, fingerprints can be located more effectively in the index.

The below figure 5 illustrates When transmitting the chunk from the client machine, the fingerprints of the pieces are compared locally. If the fingerprint is found in the local index, the data chunks and fingerprints are not sent to the cloud. As a result, data chunks with identical fingerprints are deemed duplicates and are not transferred to the cloud. Data chunks with fingerprints that are not discovered in the global index they are regarded new chunks, and their fingerprint is saved as a new iteration in the global index. This approach prevents duplicate data chunks from being sent from the client to the cloud. By avoiding delivering duplicate pieces, the bandwidth is properly used.

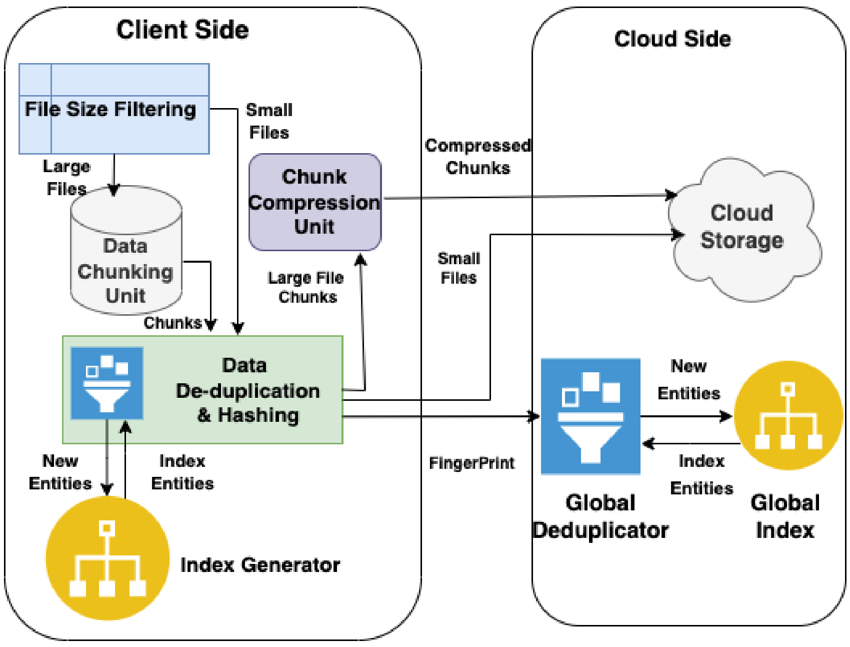


Figure 5: Architecture of Cloud Configuration

5 Evaluation

In this section we will be evaluating the various datasets on 2 given parameters to differentiate the performance of our algorithm on the serial environment and in case of scalable. We will be deploying these on Cloud architecture for better results.
The parameters on which we will be evaluating are calculated on the basis of given terms:

5.1 Time Complexity for Parallel Approach

The time complexity is dependent on all the 4 stages mentioned before and categorised under pipeline, indicated by $Time_c$, $Time_f$, $Time_i$, $Time_s$ and The fraction (the ratio of size of data prior to actually deduplication) is denoted below D_r and T_s/D_r signifies the duration of producing the quantity of non redundant entities.

Without dedupe, average running time may be approximated as follows:

Throughput = $1/Time_s$

With classic single threaded deduplication, the write throughput may be computed as follows:

Throughput = $1/(Time_c{+}Time_f{+}Time_i{+}Time_s/D_r)$

When the chunking and fingerprinting operations are further parallelized with parallel threads (N), the throughput is as follows:

Throughput = 1-Value of Maximum($Time_c$/N, $Time_f$/N, $Time_i$, $Time_s/D_r$)

5.2 Experimental Datasets

Chunking wastes resources and time since it must traverse the whole file byte by byte to find the cut-points. The processing time and resource usage of the chunking step are entirely dependent on the conditions of the chunking algorithms that split the file. For our experiment, we used three datasets to evaluate the method in text files of varying sizes, which will aid in establishing how effective the suggested architecture is overall represented in below Table 1

Table 1: Dataset Details

File Type	File Name	File Size
.txt	Dataset-1	29 MB
.txt	Dataset-2	38.5 MB
.txt	Dataset-3	48 MB

5.2.1 Evaluating Dataset-1

In our first experiment we are taking Dataset-1 into account to perform chunking and hashing using proposed technique.Our system will evaluate on the basis of time complexity and performance when chunking is performed on a single thread, 10 processors and 20 processors at a time. To achieve better results we are conducting 10 iteration to calculate the average time necessary to execute chunking using varying numbers of processors; as a consequence, we will conclude the total throughput for Dataset-1 in all three scenarios. The output of the same is given Table 2 and graphical representation Figure 6

Table 2: Result for Dataset-1

Chunking type	Time in Seconds	Throughput (MB/Seconds)
Serial	6.425	5.347
10 Processors	4.751	6.103
20 Processors	4.023	7.208

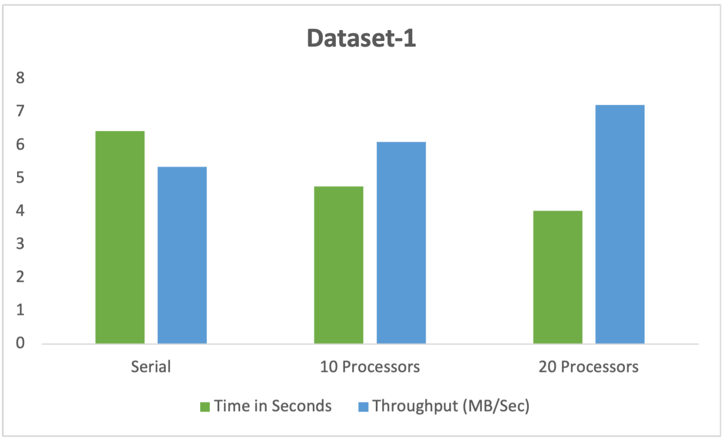


Figure 6: Throughput for Dataset-1

5.2.2 Evaluating Dataset-2

Second experiment is conducted on Dataset-2 that is of size 38.5MB, both chunking and hashing stages will go through the multi processing steps and the throughput for the same will be calculated based on the actual time taken for chunking and its throughput value. The output is represented in a Table 3 and Graph Figure 7

Table 3: Result for Dataset-2

Chunking type	Time in Seconds	Throughput (MB/Seconds)
Serial	8.119	4.741
10 Processors	7.121	5.406
20 Processors	5.578	6.902

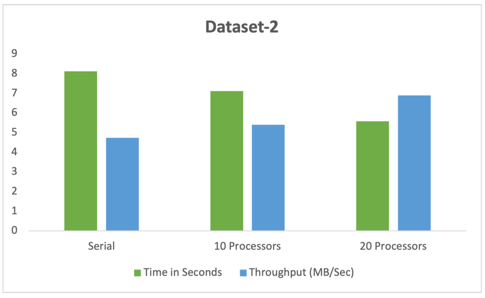


Figure 7: Throughput for Dataset-2

5.2.3 Evaluating Dataset-3

Lastly, we will be using Dataset-3 of size 48MB that is the largest amongst all, we will the executing it under same condition of different number of processors and evaluate the throughput accordingly. We have given it in below Table 4 and Graph Figure 8

Table 4: Result for Dataset-3

Chunking type	Time in Seconds	Throughput (MB/Seconds)
Serial	9.069	5.292
10 Processors	7.861	6.106
20 Processors	5.012	7.951

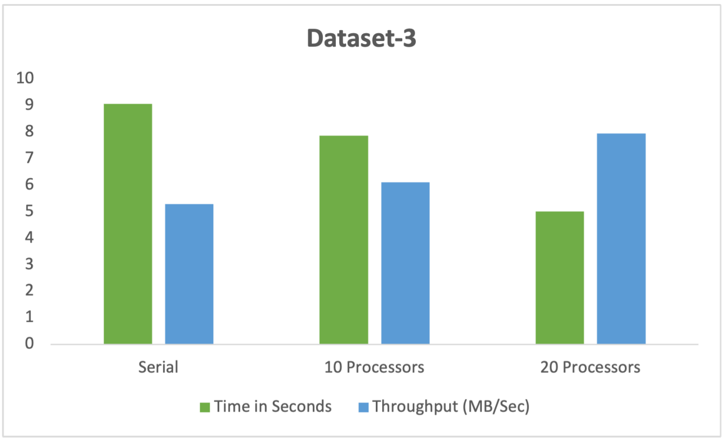


Figure 8: Throughput for Dataset-3

5.3 Duplicate Elimination Ratio (DER)

It is an important metric to analyse any chunking technique and it is represented by a fraction of original size of input stream by the resulted size of the chunked file after performing de-duplication. Its is represented like below:

$$Deduplication\ Elimination\ Ratio\ (DER) = \frac{Input\ Datas\ size\ before\ Dedulplication\ in\ MB}{Output\ Data\ size\ after\ Deduplication\ in\ MB}$$

5.3.1 De-duplication Ratio

In this part, we examined the size of all datasets after completing the de-duplication technique, comparing the real size of the datastream to the size of the file acquired after removing duplicate chunks. This computation results in the value of DER.
We examined the size of datasets in MB and evaluated the DER value associated with these sizes in the Table 5 below; the graphical representation Figure 9 makes it more

evident.

Table 5: File Size comparison based on De-duplication

Dataset	Input Datastream Size (MB)	De-duplication Size(MB)	DER
Dataset-1	29	6.5	4.461
Dataset-2	38.5	7.8	4.982
Dataset-3	48	9.6	5.135

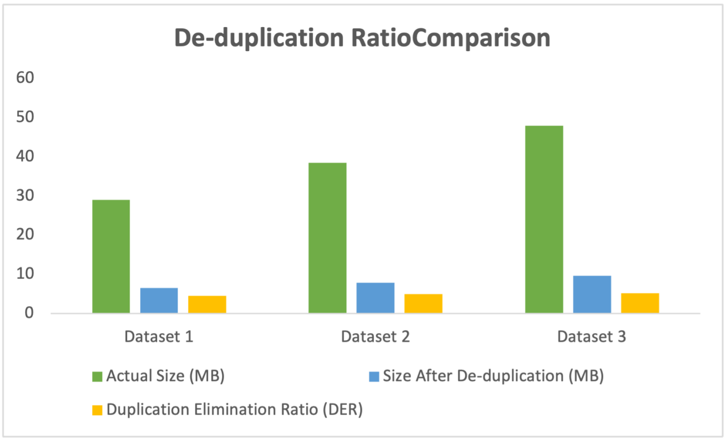


Figure 9: DER Evaluation

6 Conclusion and Future Work

6.1 Conclusion

Data Deduplication is a methodology widely used for managing efficient of storage services in a cloud environment. In this paper we would like to propose a chunking approach that pivots on byte distribution in the datastream by considering every byte while matching the hash value of the designated chunk. We focused on chunking performance by introducing the notion of multi-threads. This approach expands the chunking and hashing operations accord several processing threads, and all portions of the datastream would run parallely. The evaluation is performed on the basis of Chunking Throughput and DER using 3 different datasets and the results notifies that present chunking approach adequately performs on CDC technique with an appropriate outcome for Duplication ratio where the size of the file is reduced by 70% and the throughput of the overall chunking has improved as equivalent to the number of processors being used. While carrying out this experiment, we took the cloud environment into account in order to better understand network bandwidth utilization and the impact on cloud performance.

6.2 Future Work

In the future, we will improve chunking performance with regard to huge files that comprise media files, optimize the software based on multi-media datasets, and assess the chunking process using a multi-thread system to increase throughput and network capacity. The use of GPU to boost system speed and compress data for storage is one element to consider while working on parallel techniques for chunking and fingerprint computation.

References

Bjørner, N., Blass, A. and Gurevich, Y. (2010a). Content-dependent chunking for differential compression, the local maximum approach, *Journal of Computer and System Sciences* **76**(3-4): 154–203.

Bjørner, N., Blass, A. and Gurevich, Y. (2010b). Content-dependent chunking for differential compression, the local maximum approach, *Journal of Computer and System Sciences* **76**(3-4): 154–203.

Eshghi, K. and Tang, H. K. (2005). A framework for analyzing and improving content-based chunking algorithms, *Hewlett-Packard Labs Technical Report TR* **30**(2005).

Gholami Taghizadeh, R., Gholami Taghizadeh, R., Khakpash, F., Binesh Marvasti, M. and Asghari, S. A. (2020). Ca-dedupe: Content-aware deduplication in ssds, *The Journal of Supercomputing* **76**(11): 8901–8921.

Guo, F. and Efstathopoulos, P. (2011). Building a high-performance deduplication system, *2011 USENIX Annual Technical Conference (USENIX ATC 11)*.

Kaur, R., Chana, I. and Bhattacharya, J. (2018). Data deduplication techniques for efficient cloud storage management: a systematic review, *The Journal of Supercomputing* **74**(5): 2035–2085.

Krishnaprasad, P. and Narayamparambil, B. A. (2013). A proposal for improving data deduplication with dual side fixed size chunking algorithm, *2013 Third International Conference on Advances in Computing and Communications*, IEEE, pp. 13–16.

Kruus, E., Ungureanu, C. and Dubnicki, C. (2010). Bimodal content defined chunking for backup streams., *Fast*, pp. 239–252.

Liu, C., Xue, Y., Ju, D. and Wang, D. (2009). A novel optimization method to improve de-duplication storage system performance, *2009 15th International Conference on Parallel and Distributed Systems*, IEEE, pp. 228–235.

Ma, J., Zhao, B., Wang, G. and Liu, X. (2012). Adaptive pipeline for deduplication, *2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, IEEE, pp. 1–6.

Muthitacharoen, A., Chen, B. and Mazieres, D. (2001). A low-bandwidth network file system, *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pp. 174–187.

Ni, F. (2019). *Designing Highly-Efficient Deduplication Systems with Optimized Computation and I/O Operations*, PhD thesis, The University of Texas at Arlington.

Nie, J., Wu, L. and Liang, J. (2019). Optimization of de-duplication technology based on cdc blocking algorithm, *2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, IEEE, pp. 1–5.

Venish, A. and Siva Sankar, K. (2016). Study of chunking algorithm in data deduplication, *Proceedings of the International Conference on Soft Computing Systems*, Springer, pp. 13–20.

Wei, J., Zhu, J. and Li, Y. (2014). Multimodal content defined chunking for data deduplication, *Huawei Technologies* .

Xia, W., Zou, X., Jiang, H., Zhou, Y., Liu, C., Feng, D., Hua, Y., Hu, Y. and Zhang, Y. (2020). The design of fast content-defined chunking for data deduplication based storage systems, *IEEE Transactions on Parallel and Distributed Systems* **31**(9): 2017–2031.

Yu, C., Zhang, C., Mao, Y. and Li, F. (2015). Leap-based content defined chunking—theory and implementation, *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*, IEEE, pp. 1–12.

Yuan, J. and Yu, S. (2013). Secure and constant cost public cloud storage auditing with deduplication, *2013 IEEE Conference on Communications and Network Security (CNS)*, IEEE, pp. 145–153.

Zhang, Y., Feng, D., Jiang, H., Xia, W., Fu, M., Huang, F. and Zhou, Y. (2016). A fast asymmetric extremum content defined chunking algorithm for data deduplication in backup storage systems, *IEEE Transactions on Computers* **66**(2): 199–211.

Ziv, J. and Lempel, A. (1978). Compression of individual sequences via variable-rate coding, *IEEE transactions on Information Theory* **24**(5): 530–536.