International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 16, No.1 (Jul-24)

http://dx.doi.org/10.12785/ijcds/160105

Deduplication using Modified Dynamic File Chunking for Big Data Mining

Saja Taha Ahmed[1]

[1]*Ministry of Education, Vocational Education Department, Iraq*

Received 19 May. 2023 , Revised 27 Feb. 2024 , Accepted 6 Apr. 2024 , Published 1 Jul. 2024

Abstract: The unpredictability of data growth necessitates data management to make optimum use of storage capacity. An innovative strategy for data deduplication is suggested in this study. The file is split into blocks of a predefined size by the predefined-size DeDuplication algorithm. The primary problem with this strategy is that the preceding sections will be relocated from their original placements if additional sections are inserted into the forefront or center of a file. As a result, the generated chunks will have a new hash value, resulting in a lower DeDuplication ratio. To overcome this drawback, this study suggests multiple characters as content-defined chunking breakpoints, which mostly depend on file internal representation and have variable chunk sizes. The experimental result shows significant improvement in the redundancy removal ratio of the Linux dataset. So, a comparison is made between the proposed fixed and dynamic deduplication stating that dynamic chunking has less average chunk size and can gain a much higher deduplication ratio.

Keywords: Fixed Chunking, Dynamic chunking, Big Data , Data Mining

1. INTRODUCTION

The economic industry in the twenty-first century relies heavily on data, a trend known as the information age. To be useful, how much data must be processed? According to a study done by IDC, only 0.5 percent of globally generated data is evaluated [1],[2]. Global Datasphere, according to the International Data Corporation (IDC Report, 2020), is the collection of data generated, acquired, or replicated via digital material from around the world. According to IDC, the Global Datasphere will rise from 33 Zettabytes (ZB) in 2018 to 175 ZB by 2025[3],[4]. Data deduplication is therefore referred to as a method for avoiding storing and transmitting redundant data over the internet that is both space and bandwidth-efficient. It is one of the best strategies for resolving this problem [5],[6].

Deduplication is an effective storage-saving method used in enterprise backup setups. The same data block might be saved numerous times on a file system across various files. Data is only stored once, enabling the use of the same distinct data across all files that employ the same portion. The most widely used technique partitions file data into pieces and creates a distinct cryptographic fingerprint for each piece. If the fingerprint has previously been saved, a chunk is classified as redundant; otherwise, it is regarded as unique [7]. The division of the input file into different segments is the most basic aspect of data deduplication. Fixed-size (FSC) and variable-size (CDC) chunking are the two most prevalent algorithms for chunking data [8],[9].

The simplest and quickest chunking method is fixed-size. Although most of the data in the file are unaffected, the boundary shifting issue reduces the effectiveness of deduplication; FSC may provide various outcomes (i.e., various hash) for all subsequent chunks. All existing chunk boundaries indicated by FSC are altered if a single byte is inserted at the beginning of a data input stream, and duplicate chunks are not found and eliminated [1]. Content-Defined Chunking (CDC), another name for variable-length chunking, is a recommended method for resolving the boundary shifting issue [11], in which, unlike fixed-size chunking, chunk boundaries are determined by the information of the data stream in bytes. Consequentially, when data is modified, the majority of the chunks stay unaffected, allowing for the detection of additional redundancy for deduplication [12], [13]. According to several recent studies, around (10–20)% more duplication can be detected by CDC methods than by FSC[14],[15].
Deduplication is a vital step in the data preprocessing phase of data mining that helps integrate data from many sources. The redundant and/or inconsistent data for text/data mining applications, might distort data distribution and result in an imbalance. Developing a system that can predict two records having the same content despite multiple data inconsistencies is an important issue faced by this work [16].

The deduplication technology locates and removes duplicated data blocks using a hash function. Hashing al-

E-mail address: sajataha@ymail.com, drsajataha@gmail.com

gorithms are used in data deduplication solutions to detect discrete ”chunks” of data. There are two widely employed procedures: SHA-1 and MD5. When an algorithm for hashing analyses data, it creates a hash that serves as a representation of the information and uses various comparison techniques to identify redundancies. The same hash is produced each time the same data is sent through the hashing process multiple times. On the other hand, deduplication has a lengthy runtime and requires more processing power [17].

This article describes a deduplication method that breaks the byte stream into parts in an attempt to discover differentiating signatures that can be used across chunks. The suggested deduplication system is implemented with the following contributions to the paper:
1. A chunking approach is suggested as a multiple-character fingerprint. Identifying the most redundant cut point is used to improve and streamline the chunking strategy without hash assessment.
2. The suggested hashing function combined three hashes computed using different keys each time into one final hash to lower the possibility of a hash collision during the matching stage. These simple hashes require less storage space.
3. The effectiveness of the suggested strategy is explained by comparing the proposed dynamic and fixed chunking.

The proposed system is considered throughout the original paper. Section 2 describes several data deduplication-related works. The proposed system structure is defined in Section 3. The findings of the suggested technique are addressed in Section 4. The final section offers some observations as well as some suggestions for further study.

2. RELATED WORK

Duplicate data causes many problems with search and security, taking up more space and lengthening access times. Such issues can be effectively solved through data mining. Deduplication development studies have been conducted by researchers from a wide range of perspectives, mostly focused on data partitioning (extraction of features) methodologies and the formation of chunk signatures.

KRISHNAPRASAD, et al (2013) [18] introduced a strategy termed ”Dual Side Fixed Size Chunking”, by creating a hash from both the start and the end of the file and saving both values to a metadata table. The suggested approach successfully addresses the fundamental drawback of fixed chunking, namely the boundary shifting issue, without the need for expensive calculations for variable-size chunking, or content-defined chunking. This approach can be used to create a better DeDuplication ratio for video or audio files. However, this algorithm required more data storage to store chunks from frontend and backward files. To tackle the boundary shift problem of fixed chunking, CDC was developed by LU, et al (2018) [19]. The use of CDC for network block storage deduplication has two challenges, though. One challenge is establishing a mapping mechanism between a deduplicated chunk’s stream offsets and its block address; another is designing an efficient index structure to handle the metadata of data chunks. This study designed two structures B+ trees and hash tables to solve the mapping problem and provide two backends to store metadata on network block storage devices. to accelerate disk searches by lowering the volume of the hash table and the lookup range. The proposed systems are evaluated concerning actual workloads. The experimental findings demonstrated the high search performance of the suggested schemes at a reasonable cost in terms of spatial sacrifice.

By deciding on the best chunking combination, fingerprinting, and hash table techniques, YOON (2019)[20] introduced the implementation methodology for packet-level deduplication. The results showed that performance is enhanced three times over the existing approach and that variable-size chunking greatly decreased redundancy than fixed-size chunking, even though fixed-size chunking was faster. XU, et al. (2019)[21] proposed LIPA, a learning-based data deduplication system that uses the reinforcement learning paradigm to create a flexible indexing structure. To address the lookup disk bottleneck issue, it is different from past inline chunk-based deduplication algorithms for large-scale backup. In earlier techniques, chunk identification needed either a full chunk index or a sampling chunk index. The sampled chunk index has a direct impact on the deduplication ratio, which is correlated with the sampling ratio, and the entire chunk index uses a lot of RAM. The suggested learning-based strategy offers a deduplication ratio that is on par with or better than preceding techniques while consuming very little memory to keep the index.

YE, et al. (2021)[22] suggested CARD, a novel chunk-context-aware similarity detection technique that used an N-sub-chunk shingles-based initial feature extraction strategy and a BP-Neural network-based chunk-context aware model. It efficiently linked the underlying structure of each data chunk’s content with context information for feature extraction, small modifications in data chunks have a much lower impact. When compared to state-of-the-art likeness detection algorithms, the results demonstrated that CARD can discover up to 75.03 percent more duplicated data and accelerate resemblance detection procedures by 5.6 to 17.8 times.

The main goal of Data Deduplication was to optimize redundancy through analysis and updating of the current content dynamic chunking parameters via the introduction of a Novel-Hash function by Madan, S. (2022)[23], which will further enable effective chunk breakpoint recognition and dataset fingerprinting. In addition, the multithreading of content-defined chunking algorithm will be produced in this study work to improve the computing process by utilizing the multiprocessor technique. The proposed algorithm was based on the idea of a shifting window since if there is no match with the hash value pool, it slips one byte at a time. The research was comparing the average processing time for

TABLE I. The Proposed Method and Previous Work Comparison

Research	Advantage	Disadvantages	The proposed method
KRISHNAPRASAD, et al (2013) [18]	It solves the boundary-shifting issue and gains better DR	More data storage was needed for the algorithm to save chunks from the frontend and backend files	Solve the boundary shift problem with better DR without needing for double rolling file with less storage size
LU, et al (2018)[19]	It provides high search performance.	Designed two structures of B+ trees and hash tables to solve the mapping problem	Used one structure index table only to store hash value
YE, et al. (2021) [22]	Uses chunk-context aware model to detect duplications.	Used BP-Neural network for feature extraction which is time-consuming.	It uses simple DCA calculation and does not depend on ML algorithms for detecting internal representation.
Liu X, et al (2023)[24]	Uses active learning to solve deduplication issues.	Small datasets, not large data, are used in the study, and the deduplication ratio for each dataset is not evaluated	The proposed method calculates the size of the large dataset both before and after the redundant data is removed, as well as the deduplication ratio.
Guo S. (2023) [26]	It uses DSW and Markov prediction to find cut points and novel DR on Ocean datasets.	It was needed to resolve the DSW optimization operation when dealing with normal data. Furthermore, the largest dataset weighed in at under 2 GB. Only one technology was used for the experiments.	The suggested approach was tested on a second device that has other specifications with a dataset larger than 2 GB, irrespective of the type of data.

a file between the parallel environment and the serial way, and evaluating the system on the AWS cloud infrastructure with various datasets. The research results show that the proposed method increases storage efficiency by 70% while reducing execution time.

A pre-trained deduplication model based on active learning to solve the issue of data duplication is proposed by Liu X, et al. (2023) [24]. A Transformer that was tuned to manage how the deduplication issue appears in the classification process was used in the development of the framework. The Transformer has already undergone training. The Rdrop technique was utilized in the deduplication model training to perform data augmentation on each iteration of labeled data, hence reducing the cost associated with human labeling. According to experimental results, the developed performs up to 28% better on benchmark datasets in Recall than the previous state-of-the-art for deduplicated data identification.

Revathy, S., and D. Viji (2023)[25] proposed a system that aims to reduce the storage space using a new deduplication technique by employing hash indexing forums. Sparse augmentation is the first preprocessing method used. To create a hash index, the preprocessed files are further divided into blocks. Related files are clustered together once the similar blocks are identified using Semantic Content Source and Distance Vector Weightage Correlation, which calculates the document similarity weight. By testing 5 GB of data, the suggested system achieves great performance with a precision rate of up to 89.7%, and its recall rate is 87.6% better than that of alternative techniques.

The segmentation technology proposed in this research by Guo S. (2023) [26] was based on dual sliding windows. The double sliding window (DSW) structure reduced the amount of memory used by the fingerprint table by dividing the data size of blocks into more typical pieces. To increase cutting efficiency, researchers also incorporate an algorithm for forecasting into the reduction system (Markov prediction-based approach) to forecast the data block's cutting point. In addition, the paper suggested a novel strategy for figuring out the deduplication ratio. The experimental results showed that BSW (Basic Sliding Window) performs poorly when dealing with ocean data, so DSW outperforms state-of-the-art algorithms (BSW, TTTD, FSC) in this regard and has a maximum restriction on the block size.

To demonstrate the benefits and drawbacks of the proposed technology in contrast to earlier research technologies. Table 1 displays the suggested system compared to prior research efforts.

3. THE PROPOSED SYSTEM

The dataset for the proposed system includes a variety of files of varying sizes and types. The system processes each of these files one at a time. To show the efficiency of the suggested method, three Linux file system datasets of

versions 4, 5, and 3, respectively, were used to construct and evaluate the proposed method. Chunking, hashing and indexing, and matching are the three stages of any deduplication system.

The proposed deduplication systems break files into chunks and compare chunk data to detect duplicated segments. Many features are required to maintain the relationships between files and chunks, which necessitate additional resources beyond the deduplicated data. The chunk index holds the chunk information for the chunks that have been saved. Every deduplication system has a permanent index that stores the information needed to reconstruct file contents using file recipes. A list of chunk identifiers can be found in a file recipe. Each of these chunk identifiers is a unique identifier for a certain chunk. The file recipe and the uniquely identifiable chunked data can be used to reconstitute the original file contents (referred to as logical data). The chunk identifiers are read, and their related data chunks are loaded and concatenated in a specific sequence, to recreate the logical data. The initial stage of the deduplication process is chunking. It works by dividing the income data flow (i.e., file) into distinct, independent chunks using a repeating hash. A chunking procedure searches the file for the dataset's sequence of bytes' highest probable hash value, which is then used as a cut point. There are two types of chunking algorithms proposed:

A. *The proposed Dynamic Chunking Algorithm (DCA)*

The suggested deduplication technique divides files into chunks depending on the file's most well-established patterns regarded as file breakpoints or divisors. To put it another way, cut points represent some internal aspects of the files. The suggested approach takes advantage of the chunk's highest-probability bytes, which are sometimes located at the chunk's boundary and/or included in the chunk. The recommended bytes are utilized to set the breaks rather than utilizing arbitrators based on a criterion value for per rotating hash calculation, which wastes resources (i.e., typical Rabin fingerprint).

This phase's main objective is to divide the incoming file into manageable, nonoverlapping chunks using the DCA technique. It will decide on the most common divisors to define the chunk boundary or breakpoint based on the dataset's contents. In addition to using the original file's characteristic, the frequency of double bytes in the dataset is used to establish the borders. The criterion is used in DCA to either discover the cut point after the minimum threshold value or to reach the maximum threshold value if a determination of the breakpoint using the cut point criteria fails. A minimum and maximum chunk size are promised by DCA. Table 2 shows the DCA Parameters' Implications.

The deduplication methodology can offer the best duplication detection ratio whenever the chunking method is dependent on the file format if compared to fixed / variable-size chunking methods. The suggested DCA algorithm is shown in Figure 1, and the Chunking steps are as follows:

TABLE II. The Benefits Of DCA Variables

The variable	Benefits
Min th	To control very small chunks
C	To identify chunks' divisors
Max th.	To control very big chunks

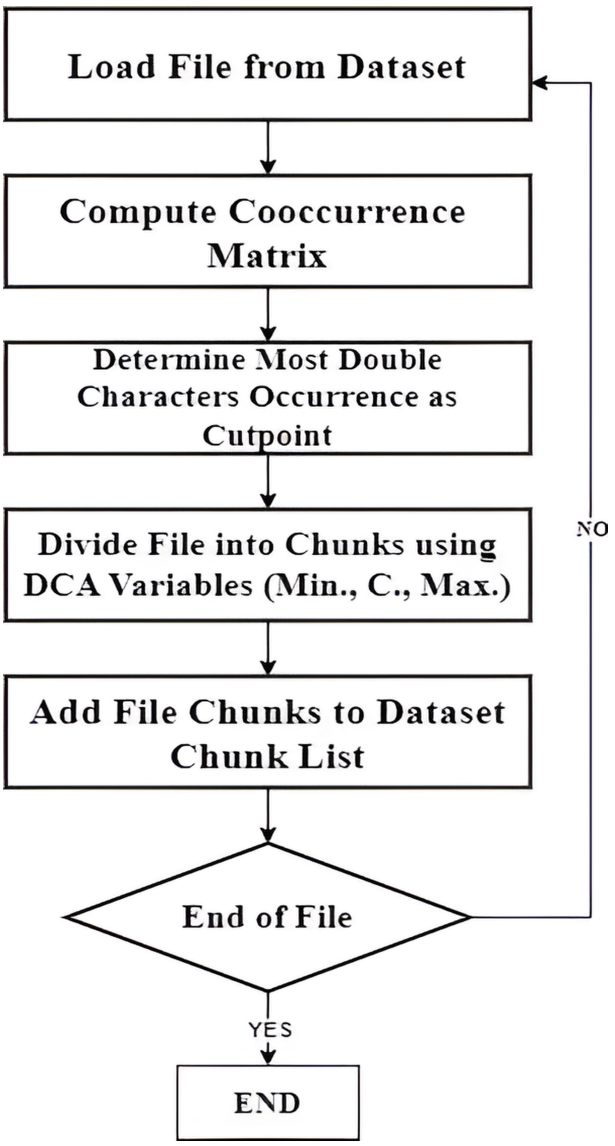

Figure 1. The Dynamic Chunking Algorithm (DCA)

Algorithm 1. Dynamic Chunking Algorithm

Objective: Split the file into sections using Min Th. and Max Th., considering only double-byte appearances.
Input: An array of bytes representing a file with various sizes and types.
Output: Various sizes of chunks.
Step1: Do this for each dataset file.
Set Len to Length of File
Compute the cooccurrence matrix between every two bytes in a file
Select two bytes that have maximum occurrence in the matrix as file cutpoint
Set div1 to byte1
Set div2 to byte2
Step2: While Not File's End, Do
If Len = 0 ; Go to Step 3
Else If Len ¡ = Min Th.
Add file to Chunk list Go to Step 3
Else
Set the chunk to the file's bytes from 0 to Min Th.
Set count to Min th.
While counting less than file length
if (count equal to file Length - 1)
add a chunk to Chunk list, break
Set ch1 to file[count]
Set ch2 to file[count+1]
if (chunk Length greater than or equal to Max th.)
add chunk to Chunk list, set chunk to null // add old chunk and create new one
if ((ch1 not equal to div1) && (ch2 not equal to div2))
insert new byte from file to chunk
else
if (chunk Length less than or equal to Min th.) insert new byte from file to chunk
add char until reach Min th.
else
add a chunk-to-Chunk list, set chunk to null
add old chunk and create a new one
Step3: Return file Chunk list
Step4: add file chunk list to dataset chunk list, Go to step1
Step5 : End

Algorithm 2. Fixed Chunking Algorithm

Objective: Depending on the file size, divide the file into equal portions.
Input: An array of bytes representing a file with various sizes and types, size ofchunks.
Output: Chunks of fixed sizes.
Step1: Step 1: for each file in the dataset do
Set Len to Length of File
Set N to the size of chunks
If Len = 0 Go to Step 1
Set counter to 0
Step 2: While Not File's End, Do
If (counter+N¡= len)
Set chunk to bytes of file from counter to N+counter
add a chunk to the Chunk list, set the chunk to null
add the old chunk, and create a new one
set counter to counter+N
else
to add the tail of a file
Set file's chunk to bytes, from counter to Len
add a chunk to the Chunk list, set the chunk to null
Step 3: Return file Chunk list
Step 4: : add file chunk list to dataset chunk list, Go to step1
Step 5:End

B. *The Proposed Fixed Chunking Algorithm (FCA)*

The suggested system breaks the file into equal chunks as fixed-size blocks have typically been used in file systems to split a byte sequence into blocks. Although it provides a simpler implementation and is extremely fast, the fundamental flaw with this approach is that a single byte introduced or removed in the sequence will cause all block borders to migrate, resulting in different blocks. As a result, following every insertion or deletion to the file, different blocks would be produced. This is far from the desired stability when using the local modification attribute.

The proposed FCA divides files into equal chunks based on their size, improving the deduplication ratio significantly, though not as much as dynamic content chunking. The file length must be examined to ensure that it is long enough to be divided into several equal chunks. Algorithm 2 depicts the suggested Fixed Chunking Algorithm. Furthermore, Figure 2 shows a difference in chunking between DCA and FCA algorithms.

C. *The Proposed Duplication Data Prediction*

Traditional deduplication systems waste time trying to address the collision problem and need a lot of processing power and storage. This study suggests the X-hashing strategy to conserve resources and speed up processing, where X is number of hashing function in this paper X equals three. Traditional hashing functions (SHA1 and MD5), which are utilized by other content-defined chunking methods, demand a lot of computing resources to calculate hash values and a lot of storage space to store them. The number of bits required to save our proposed hashes is smaller than the number of bits required to save SHA1 (160 bits) and MD5 hash values (128 bits). Using the hash functions indicated below, the approach can compute and preserve one hash value for each chunk based on three hashes computed previously using different keys for the same chunk:

$$H_1(C) = \sum_{i=0}^{L} \sum_{J=0}^{N} (Ch[i] \oplus k_1[j]) \quad (1)$$

$$H_2(C) = \sum_{i=0}^{L} \sum_{J=0}^{N} (Ch[i] \oplus k_2[j]) \quad (2)$$

$$H_3(C) = \sum_{i=0}^{L} \sum_{J=0}^{N} (Ch[i] \oplus k_3[j]) \quad (3)$$

$$FinalHashValue = H1||H2||H3 \oplus 1's(48Bits) \quad (4)$$

Since hash computing process involves two main steps 1-**Key Generation** : the preprocessing step involves generating k1, k2, and k3 are three Max Th. Arrays (rows) of random numbers. The created rows are employed to generate a series of integers once and store it for use in equations 1, 2, and 3.
2- **Hash Generation**: where L is the size of a binary chunk array and Ch is an array of binary chunk bytes, and N is equal to Max Th. The XOX Function is used that produce a 48-bit message digest value.

The proposed system can be implemented on any network (clients and servers), and on each side, the incoming file is divided into chunks according to DCA or FCA. The final hash for the incoming chunk is compared with preserved hashes in the index table. The distinct data container preserves the incoming (non-duplicated) chunk if there isn't a matching chunk hash. The chunk identifier is recorded in the database, and the final hash value is kept in a hash table for any further computation. If an incoming

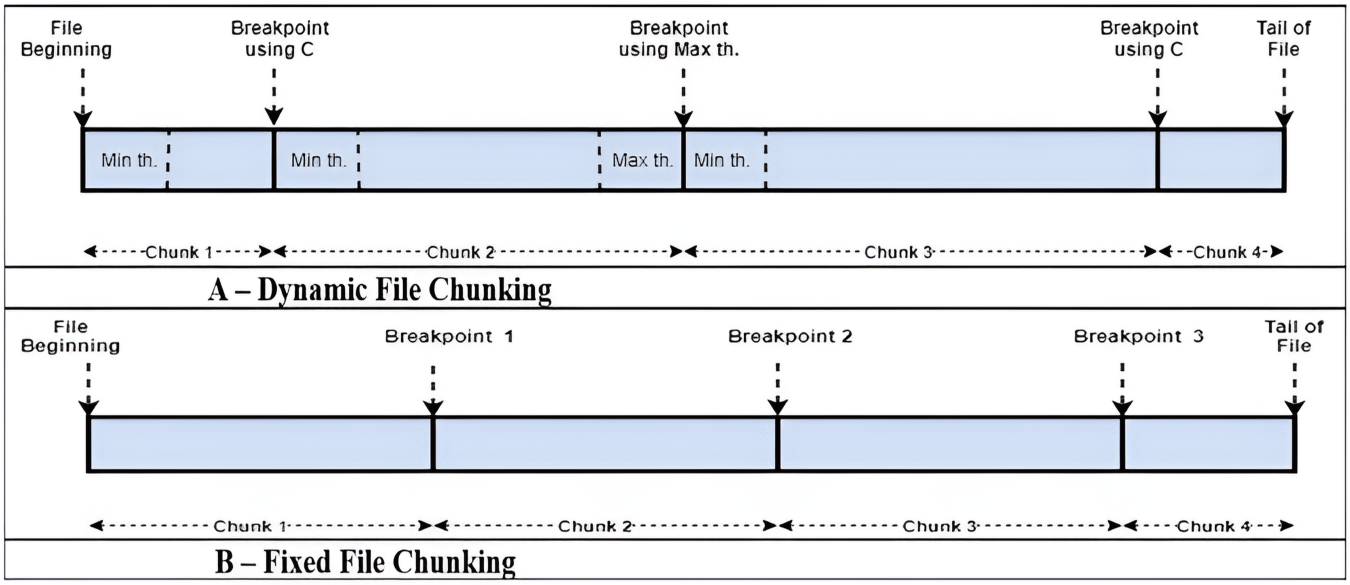

Figure 2. Dynamic vs. Fixed File Chunking

chunk hash is redundant, on the other hand, the chunk's data table pointer is changed to the current chunk, and the incoming (replica) chunk is released by raising the chunk pointer count for that chunk. The amount of time needed to match the byte-to-byte chunks will be reduced by this comparison.

4. EXPERIMENTAL RESULTS

The conducted results were obtained by performing accurate and equitable assessments. In this paper, we created a deduplication storage system and assessed how well the suggested algorithms performed. In order to compare the behavior of the proposed system, the test outcomes are evaluated using the following performance metrics

1- Equation 5 is used to express the Deduplication Ratio (DR), which assesses the efficiency of the deduplication procedure [27][28].

DR= (Total Input Data Size Before Deduplication) /(Total Input Data Size After Deduplication) (5)

2- Calculate the Average Chunk Size (AVG) by dividing the total data size by the number of segments overall [29][30].

A computer with a 64-bit Windows operating system, an Intel CORE-i7 CPU, and 16.00 Gigabytes of RAM was used to create the suggested system. In addition, C# and Visual Studio 2019 were used to put the conceptual strategy into implementation. The tests are conducted from two aspects: the first is to study and analyze the proposed system's functionality in terms of dynamic and fixed chunking. The second illustrates the suggested deduplication system's effectiveness by comparing system behavior after identifying the best parameters for both methods.

The performance of data de-duplication systems in general is significantly impacted by the chunking technique in a variety of ways. In order to discover shift boundary concerns, the suggested dynamic deduplication methodology defines the chunk borders by a file's content. As a result, the chunking strategy is significantly influenced by the proposed double characters fingerprint when it tends to use the most identical cut points across different files in the investigated dataset. The suggested system's performance metrics results are displayed in Table 3. The evaluated chunks with the best removal redundant ratio had chunk sizes ranging from 128 to 1024 bytes.

TABLE III. Dynamic Chunking with Different Min and Max Thresholds

Min	Max	All Chunk	No. Dup.	Ratio	Dataset Size following Dedup.(MB)	AVG
128	256	14957519	5854652	2.49	851	142
128	512	14801539	5740241	2.517	844	143
128	1024	14748296	5708517	2.518	843	144
128	2048	14732830	5699198	2.515	844	144
128	4096	14729442	5696623	2.514	844	144
128	8192	14728808	5696184	2.514	845	144
256	512	13945673	7687230	2.495	988	179
256	1024	13808903	7668521	2.488	986	180
256	2048	13523561	7476898	2.451	985	184
256	4096	13387328	7207395	2.421	984	186
256	8192	13189436	7115923	2.411	983	189
512	1024	12468982	9867295	2.200	1129	200
512	2048	12289735	9587624	2.221	1138	203
512	4096	12045396	9309184	2.220	1129	207
512	8192	11482650	9231980	2.221	1135	217

When chunks have a shorter content segment, the proposed algorithm can achieve a greater DR; on the other hand, adopting a long segment may not help duplicated chunks be found. The impact of min-max th. on the deduplication ratio is shown in Figure 3. Increasing the min-max section leads to the removal of frequently shared sequences among files as relevant chunks that occur over

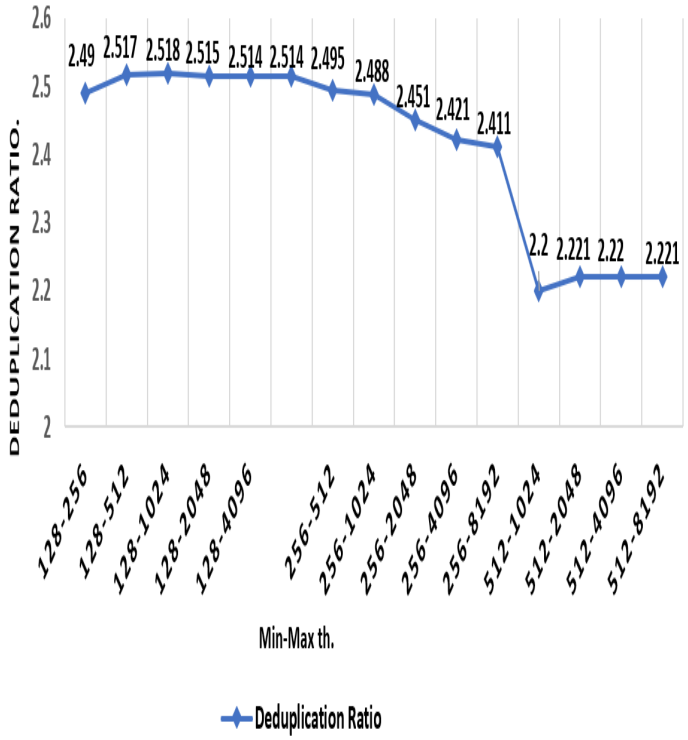

Figure 3. The Effect of Min-Max Th. on The Deduplication Ratio Using DCA

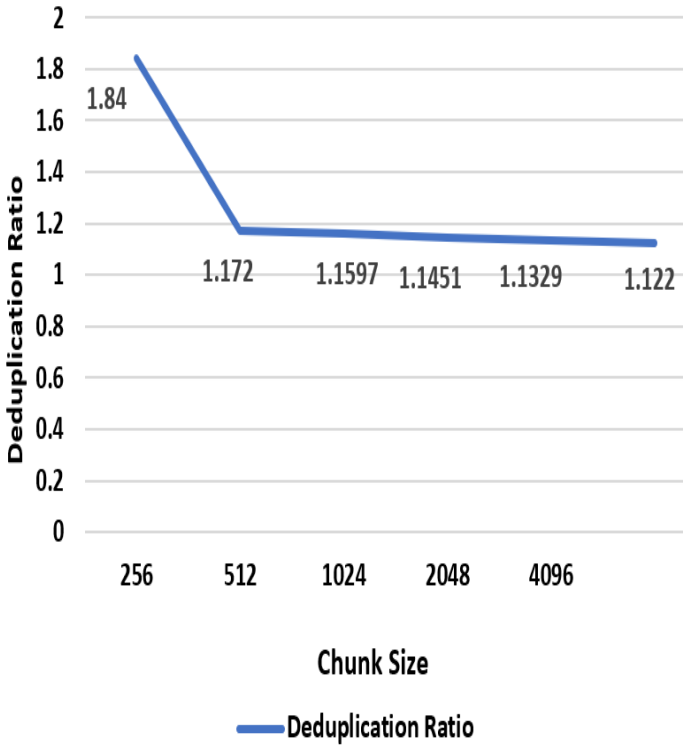

Figure 4. Impact of Chunk Size on Deduplication Ratio Using FCA

TABLE IV. Fixed Chunking with Different Chunk Size

Chunk Size	All Chunk	No. Dup.	Ratio	AVG
256	8620593	7034220	1.840	244
512	4354557	3583603	1.172	484
1024	2223128	1846807	1.159	949
2048	1160020	972700	1.145	1819
4096	633305	532385	1.132	3332
8192	376929	314780	1.122	5598

the dataset's files contain identical patterns that reflect distinctive fingerprints. The deduplication ratio of 2.518 for chunks of 128 to 1024 bytes results in the best deduplication performance.

From the beginning of the file, a fixed deduplication method works by splitting a dataset into fixed-size pieces or blocks. However, the fundamental limitation of this strategy is that if new chunks are inserted in the front or middle of a file, the existing chunks will be shifted from their original positions. As a result, the subsequent chunks will have a new hash value, resulting in a lower DeDuplication ratio. Table 4 illustrates fixed chunking performance with different chunk sizes, the best-fixed deduplication ratio is attained using chunks of 256 bytes with a ratio of 1.840. Figure 4 states the relationship of chunk size with the removal ratio of FCA. Therefore, we can approve that minimum chunk size has a better deduplication ratio.

The average size of chunks will increase if a longer threshold is considered as presented in Tables 3 and 4. In a hash-based prediction of de-duplication, increasing the chunks typically implies expanding the lookup table, which makes comparisons take longer. However, the suggested duplication prediction increases the number of small chunk lengths while incurring lower CPU overhead costs, increasing the DR with an improvement in chunking time. More redundant chunks must be removed in order to increase DR, which can also reduce the size of the lookup table.

The proposed DCA has chunks with less average size even for larger thresholds which are around from 142 to 217. In comparison to a fixed chunking algorithm, which has chunks of average size from 244 to 5598. Figure 5 and Figure 6 show the average size of chunks for the dynamic and fixed chunking algorithms, respectively. So dynamic chunking has a less average chunking size of 142 bytes for 128-256 thresholds. Because the proposed DCA leveraged the internal representation of the file to split it into segments since the contents of related segments are the same. thus, the variable-length deduplication approach (DCA) is confirmed to handle duplicated chunks better than fixed chunking (FCA).

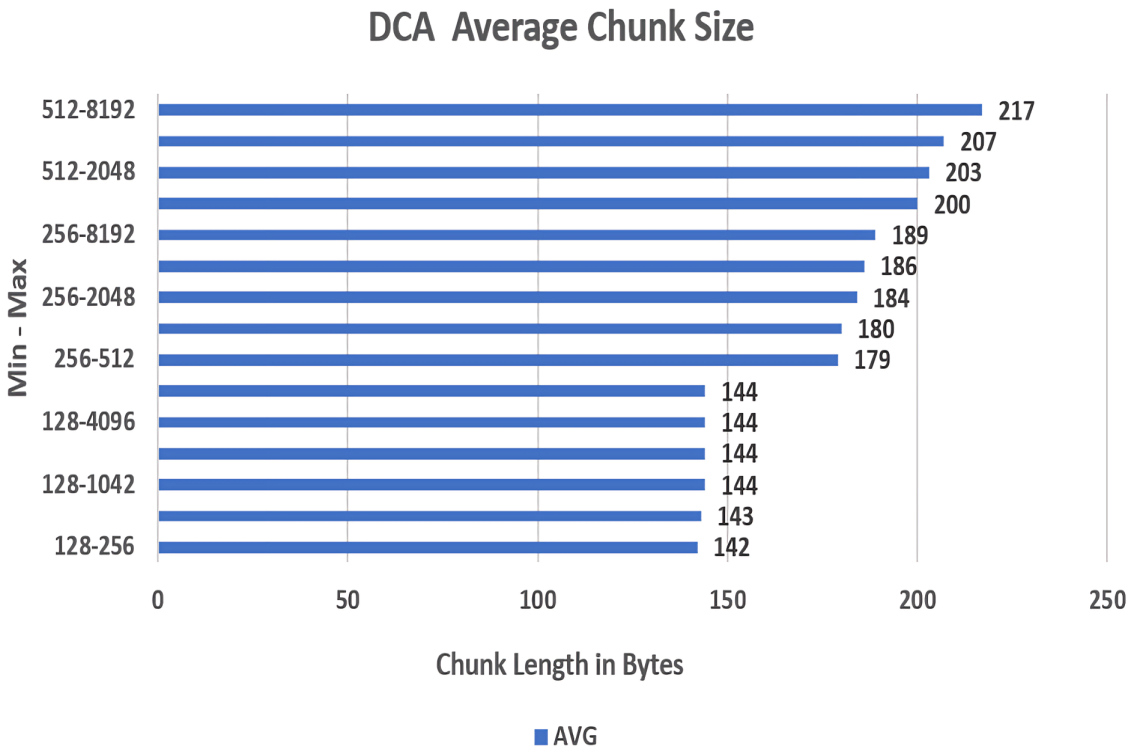

Figure 5. The Average Size of Chunks for The Dynamic Chunking Algorithm

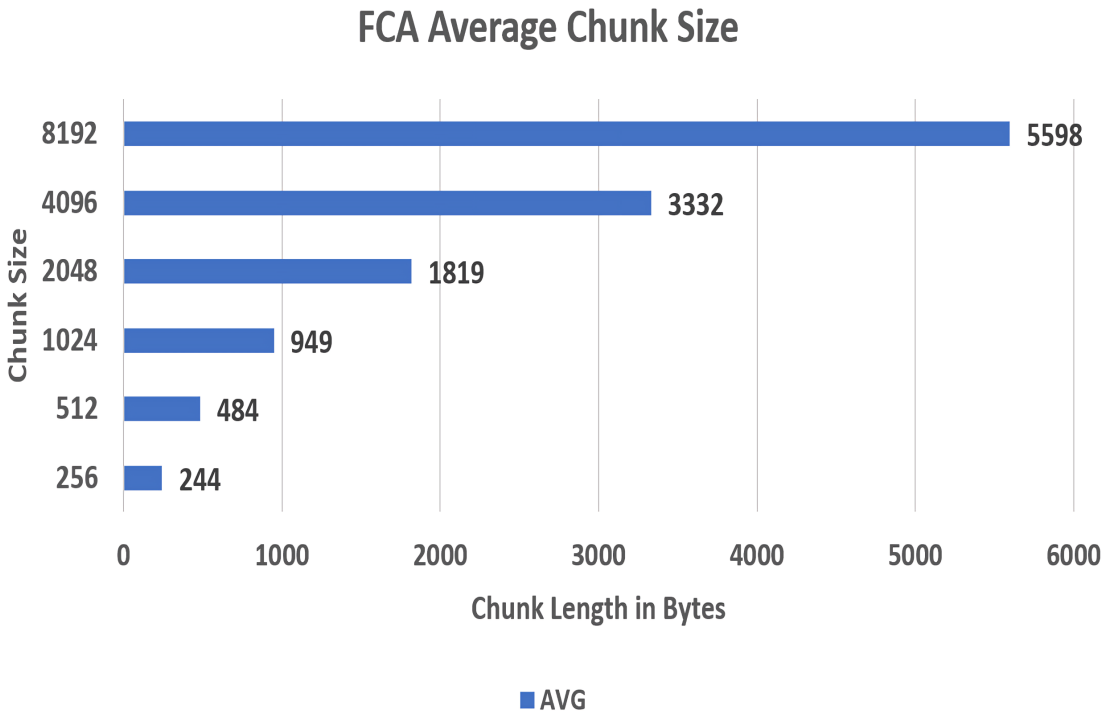

Figure 6. The Average Size of Chunks for The Fixed Chunking Algorithm

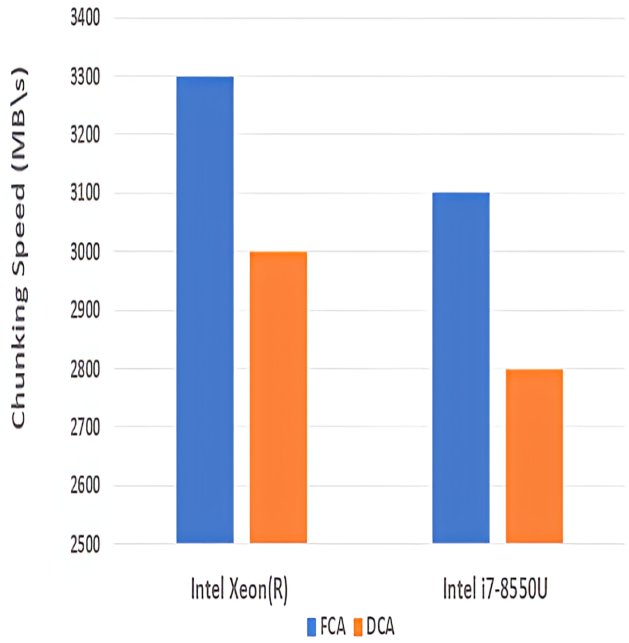

Figure 7. the Proposed FCA and DCA Chunking Speeds on Two Processors.

We configured the suggested data deduplication system on an Intel i7-8550U processor clocked at 1.8 GHz in order to evaluate FCA and DCA. To more precisely evaluate the chunking speed, a second Intel Gold 6130 CPU operating at 2.1GHz with 128GB of RAM is also employed as a benchmark processor. We consider the best-fixed deduplication ratio to be attained using chunks of 256 bytes with the ratio of 1.840 for FCA. The best deduplication performance is achieved for chunks of 128 to 1024 bytes with the DCA deduplication ratio of 2.518. Figure 7 shows that the FCA is generally faster than the DCA since it has a simpler implementation and is extremely rapid because it lacks fingerprint judgment. The two recommended algorithms' speed differences are reasonable because DCA, which uses fast key and hash generation for chunking, simplifies the hash judgment.

5. CONCLUSION

This research explores the relationships and influence of various parameters on the effectiveness of the deduplication system utilizing the proposed DCA and FCA. The combination of two innovative algorithms with the proposed hash-based prediction approach enables remarkable storage conservation by consuming less space and maximizing processing throughput. The DCA chunking algorithm is recommended for enhancing the DR and hence chunking throughput. According to the findings, the double group of bytes has a larger DR for chunking process completion. In addition, the best Min-Max Th. is 128–1024 bytes. As a result, DCA outperforms FCA in terms of DR and average chunk size.

In future studies, we'll look into dynamic allocation for specific keywords of divisors, such as using a set of divisors that may be updated over time. Furthermore, once the computer configuration is upgraded, we will study larger datasets.

Finally, a successful deduplication system must establish a proper balance between the variables that affect the system's ability to detect high levels of duplication with minimal overhead processing.

REFERENCES

[1] Meyer, D. T., & Bolosky, W. J.; "A study of practical deduplication". ACM Transactions on Storage (ToS), 7(4), 1-20., 2012.

[2] El-Shimi, A., Kalach, R., Kumar, A., Ottean, A., Li, J., & Sengupta, S.; "Primary data deduplication—large scale study and system design". In Presented as part of the 2012 USENIX Annual Technical Conference (USENIXATC 12) (pp. 285-296), 2012.

[3] Wallace, G., Douglis, F., Qian, H., Shilane, P., Smaldone, S., Chamness, M., & Hsu, W.; "Characteristics of backup workloads in production systems". In FAST (Vol. 12, pp. 4-4), 2012.

[4] Shilane, P., Huang, M., Wallace, G., & Hsu, W.; "WAN-optimized replication of backup datasets using stream-informed delta compression". ACM Transactions on Storage (ToS), 8(4), 1-26, 2012.

[5] Gantz, J., & Reinsel, D.; "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east". IDC iView: IDC Analyze the future, 2007(2012), 1-16. 2012.

[6] Kruus, E., Ungureanu, C., & Dubnicki, C.; "Bimodal content defined chunking for backup streams". In Fast (pp. 239-252), 2010.

[7] YU, Chia-Mu, et al. Privacy aware data deduplication for side channel in cloud storage. IEEE Transactions on Cloud Computing, 2018, 8.2: 597-609.

[8] Periasamy, J. K., & Latha, B.; "Efficient hash function–based duplication detection algorithm for data Deduplication deduction and the reduction". Concurrency and Computation: Practice and Experience, e5213, 2019.

[9] Sun, Z., Shen, J. & Yong, J.; "A novel approach to data deduplication over the engineering-oriented cloud systems. Integrated Computer-Aided Engineering", 20 (1), Pp. 45-57, 2013.

[10] ZHANG, Zihao, et al. SLIMSTORE: A Cloud-based Deduplication System for Multi-version Backups. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE). IEEE, 2021. p. 1841-1846.

[11] CHAPUIS, Bertil; GARBINATO, Benoît; ANDRITSOS, Periklis. Throughput: A key performance measure of content-defined chunking algorithms. In: 2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW). IEEE, 2016. p. 7-12.

[12] WIDODO, Ryan NS; LIM, Hyotaek; ATIQUZZAMAN, Mohammed. A new content-defined chunking algorithm for data deduplication in cloud storage. Future Generation Computer Systems, 2017, 71: 145-156.

[13] Xia, W., Jiang, H., Feng, D., Douglis, F., Shilane, P., Hua, Y., Fu, M., Zhang, Y. and Zhou, Y., 2016. A comprehensive study of the past, present, and future of data deduplication. Proceedings of the IEEE, 104(9), pp.1681-1710.

[14] Meister, D., Kaiser, J., Brinkmann, A., Cortes, T., Kuhn, M., & Kunkel, J.; "A study on data deduplication in HPC storage systems". In SC'12: Proceedings of the International Conference on High-Performance Computing, Networking, Storage and Analysis (pp. 1-11). IEEE., 2012.

[15] Xia, W., Zhou, Y., Jiang, H., Feng, D., Hua, Y., Hu, Y., & Zhang, Y.; "Fastcdc: a fast and efficient content-defined chunking approach for data deduplication". In 2016 USENIX Annual Technical Conference (USENIXATC 16) (pp. 101-114), 2016.

[16] SHARMA, Niteesha; PRASAD, AV Krishna; KAKULAPATI, V. Data deduplication techniques for big data storage systems. Int. J. Innov. Technol. Explor. Eng., 2019, 8.10: 1145-1150.

[17] NI, Fan. Designing Highly-Efficient Deduplication Systems with Optimized Computation and I/O Operations. 2019. PhD Thesis. The University of Texas at Arlington.

[18] KRISHNAPRASAD, P. K.; NARAYAMPARAMBIL, Biju Abraham. A proposal for improving data deduplication with dual side fixed size chunking algorithm. In: 2013 Third International Conference on Advances in Computing and Communications. IEEE, 2013. p. 13-16.

[19] LU, Hongli, et al. Efficient Online Stream Deduplication for Network Block Storage. In: 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom). IEEE, 2018. p. 111-119.

[20] YOON, MyungKeun. A constant-time chunking algorithm for packet-level deduplication. ICT Express, 2019, 5.2: 131-135.

[21] XU, Guangping, et al. Lipa: A learning-based indexing and prefetching approach for data deduplication. In: 2019 35th Symposium on mass storage systems and technologies (MSST). IEEE, 2019. p. 299-310.

[22] YE, Xuming, et al. Chunk Content is not Enough: Chunk-Context Aware Resemblance Detection for Deduplication Delta Compression. arXiv preprint arXiv:2106.01273, 2021.

[23] Madan, S. "Systemization and Evaluation for Data deduplication by deploying competitive chunking algorithm in polymorphic thread environment and avant-garde hashing techniques." PhD diss., Dublin, National College of Ireland, 2022.

[24] Liu X, Du S, Lv F, Xue H, Hu J, Li T. A Pre-trained Data Deduplication Model based on Active Learning. arXiv preprint arXiv:2308.00721. 2023 Jul 31.

[25] Viji, D., and S. Revathy. "Hash-Indexing Block-Based Deduplication Algorithm for Reducing Storage in the Cloud." Comput. Syst. Sci. Eng. 46.1 (2023): 27-42.

[26] Guo S, Mao X, Sun M, Wang S. Double sliding window chunking algorithm for data deduplication in ocean observation. IEEE Access. 2023 May 16.

[27] Luo, S., Zhang, G., Wu, C., Khan, S. and Li, K., 2015. Boafft: distributed deduplication for big data storage in the cloud. IEEE transactions on cloud computing.

[28] Thwel, T.T. and Sinha, G.R. eds., 2020. Data Deduplication Approaches: Concepts, Strategies, and Challenges. Academic Press.

[29] Ahmed, Saja Taha, and Loay E. George. "Lightweight hash-based de-duplication system using the self detection of most repeated patterns as chunks divisors." Journal of King Saud University-Computer and Information Sciences 34, no. 7 (2022): 4669-4678.

[30] Ahmed, Saja Taha "Data Mining for Non-Redundant Big Data Using dynamic KMEAN." International Journal of Computing and Digital Systems 14, no. 1 (2023): 1-1.

DR. Saja Taha Ahmed Ph.D. holder, University of Information Technology and Communication, Informatics Institute of Postgraduate Studies. MSC degree from the College of Science, Department of Computing, University of Baghdad. Higher Diploma in computer science. I am a lecturer with the vocational education department in the ministry of education.