Received: May 19, 2023. Revised: June 28, 2023.

INASS

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Enhancing Deduplication Efficiency Using Triple Bytes Cutters and Multi Hash Function

Hashem B. Jehlol[1*] Loay E. George[2]

[1]*Iraqi Commission for Computers and Informatics / Informatics Institute of Postgraduate Studies, Baghdad, Iraq*
[2]*University of Baghdad, Baghdad, Iraq*
* Corresponding author's Email: phd202020556@iips.icci.edu.iq

Abstract: Managing big data backups is challenging due to high volumes of redundant data. Data deduplication is widely used but incurs significant computational and time costs. This paper proposes a hybrid deduplication system that combines file-level and block-level methods to enhance deduplication while reducing costs. File-level deduplication eliminates duplicate files, while block-level deduplication is applied to non-duplicated files using a dynamic list of divisors to enhance deduplication. A multi-hash function generates three hash values for each file or chunk to improve chunking speed and reduce hash collisions. The proposed hybrid system outperforms other state-of-the-art methods in terms of time, data deduplication ratio, and deduplication gain. Experimental results show reductions of 97.2%, 91.6%, and 82.1% in data size for Dataset 1, Dataset 2, and Dataset 3, respectively, and demonstrate that the proposed multi-hash function is faster and requires less storage than other hash functions.

Keywords: Big data, Backup, Deduplication, File level, Block level, Hash function.

1. Introduction

Using backup storage is crucial for any data protection architecture, particularly in safeguarding users against data loss caused by accidental deletion. High backup performance is necessary when quickly backing up a significant data volume, such as big data. Due to the utilization of diverse data processing, storage, and recovery technologies in big data, there may be inherent complexities associated with these technologies [1, 2]. As the volume, variety, and speed of big data continue to grow, there is a pressing need for increased computer resources, storage space, and network bandwidth to accommodate it [1, 3]. The cost-effectiveness of extensive data storage management is a significant challenge [4]. Backup files of big data often contain redundant data due to incremental changes between backups [5, 6]; therefore, vast duplicate data across backups are frequently found [5]. Backup storage systems widely use data deduplication [7] to save storage space and achieve an impressive compression ratio and throughput performance. The deduplication process reduces the storage size, and unnecessary data, improving the efficiency of storage space utilization [8]. The backup deduplication system dramatically increased their storage efficiency [9]. Furthermore, this approach prevents the storage and transmission of duplicate data across networks [10]. Data deduplication can be performed at the file or block levels [5].

File-level deduplication involves storing only one copy of identical files, which results in reduced resource utilization and simplified implementation [11]. However, it may not detect duplicates with different names or metadata and cannot identify changes made to a single byte in the file [12]. On the other hand, Block-level deduplication can be performed using fixed or variable-length blocks. In fixed-length block deduplication, data is divided into chunks of a constant size, whereas in variable-length block deduplication, data is divided into distinct chunks based on different factors [9, 12]. While block-level deduplication is more efficient than file-level Deduplication, it requires more system resources [13, 14].

The content defined chunk (CDC) [15] technique is commonly used in data deduplication and can detect and remove duplicate data in various systems [5]. Implementing CDC improves the efficiency of the deduplication system, mainly when using the variable chunk size-based technique, which is common in backup storage systems [5]. Basic sliding window (BSW) and two threshold two divisors (TTTD) algorithms commonly used in CDC techniques rely on Rabin fingerprints of the content to identify the boundaries of chunks [16, 17]. Although these algorithms effectively detect duplicates, they can be time-consuming. This is because the Rabin fingerprints of the data stream are calculated and compared byte by byte against a condition value [18]. Furthermore, selecting appropriate values for the Threshold and divisor parameters is challenging and essential since they can significantly impact the deduplication ratio and execution time [19, 20].

Typically, data deduplication techniques consist of identifying and removing duplicate data components using a cryptographic hash function. In hash-based approaches, hashing algorithms, such as secure hash algorithm SHA1, SHA256 and message-digest algorithm MD5, generate unique hashes for each data block [21]. These hash values are then used to identify and remove duplicate data blocks during the data deduplication. The same hash value is produced when the data is processed multiple times using the hashing algorithm. However, data deduplication can be computationally intensive, resulting in long runtimes and increased processor resource usage for the system [22]. This article aims to enhance the deduplication process by improving its efficiency and increasing the deduplication ratio through three key contributions:

1. The article suggests a hybrid system integrating file-level and block-level methods to enhance deduplication. By leveraging the advantages of both file-level (lower resource utilization and simplified implementation) and chunk-level (higher efficiency) deduplication techniques
2. The article proposes a novel method for generating a dynamic list of divisors based on the frequency of triple bytes. This improves deduplication precision by accurately identifying data chunk boundaries.
3. The article presents a multi-hashing function that generates three hash values per chunk, lowering the chance of hash collisions during matching.

Also, this article improves matching processes by introducing a new method to group chunks into multiple categories before comparing them against added chunks.

2. Related work

Data deduplication techniques are heavily used in data backup. Therefore, researchers have recently considered data deduplication one of the most crucial subjects. In this paper, the following articles are considered:

H. Jasim and A. Fahad, 2018 [1] proposed a novel technique to improve the TTTD chunking algorithm called the content-based two threshold two divisor with multilevel hashing technique (CB-TTTD-multilevel hashing technique). It involves a new multilevel hashing and matching technique and a new chunk condition to generate more small chunks. They also incorporated four hashing techniques to improve the deduplication ratio and address the collision problem. However, a drawback of their method is its reliance on primitive numbers to determine the breakpoint used in TTTD. In our proposed approach, we employ a dynamic method based on the file contents to determine the divisors for the breakpoint data.

A. Saeed and L. George, 2020 [10], proposed a new method for data deduplication, the bytes frequency-based chunking (BFBC) algorithm, which outperforms commonly used CDC efficiency algorithms. The technique introduces a three-way hash function algorithm that is faster and more efficient than widely-used SHA1 and MD5. BFBC is ten times faster than BSW and around three times faster than TTTD, achieving a better deduplication ratio (DER) than other CDC algorithms. However, a notable drawback of the BFBC method is that the increased data size requires a corresponding increase in the size of the hashes used to represent the fingerprints. Consequently, this leads to an expansion of the hash index table size and imposes additional computational overhead. In our proposed approach, we address this issue by implementing file-level deduplication. By eliminating similar files at the beginning of the process, we significantly reduce the size requirements of the index table. This size reduction minimizes storage demands and alleviates the computational burden associated with managing a larger index table.

S. Ahmed and L. George, 2021 [22], Proposed a data deduplication technique that eliminates redundancy in large-scale storage by identifying cut-points in chunks using commonly repeated patterns (CRP) and a lightweight triple-level hashing function (LT-LH). The technique reduces hashing function processing and storage overhead costs. Furthermore, it is faster and more efficient than BSW and TTTD techniques. Regarding speed and storage savings, the LT-LH function outperforms SHA1 and MD5. Future

work in this study recommended dynamic allocation for the set of divisors, aiming to enhance the deduplication ratio further. In our article, A dynamic triple devisers method is developed to complement this study and improve overall deduplication performance.

Z. Xu and W. Zhang, 2022 [16] introduce QuickCDC, a technology that enhances content-defined chunking (CDC) for data deduplication. It employs three techniques to improve performance. Firstly, it jumps directly to the boundaries of duplicate chunks, skipping corresponding chunk lengths. Secondly, it skips the minimum chunk length for unique chunks. Lastly, it dynamically adjusts mask bit lengths to optimize the distribution of chunk lengths. Experimental results show that QuickCDC is 11.4 times faster than RapidCDC, improves the deduplication ratio by up to 222.3%, and increases throughput by 111.4%. Overall, QuickCDC offers significant speed improvements the data deduplication. In contrast to the mentioned study, our proposed method surpasses it by achieving significant improvements in both the speed of the deduplication process and the deduplication ratio. Our approach outperforms the referenced method regarding gain and deduplication ratio, making it a more efficient and effective solution for data deduplication.

H. Jehlol and L. George, 2022 [23] present a new method for accelerating and improving data deduplication. It classifies data based on Pearson correlation between histogram extensions, uses divisors from data repeating patterns, and introduces faster hash functions. The proposed method achieves a deduplication ratio ten times higher than the Basic Sliding Window method and approximately five times higher than the two thresholds two divisors method, demonstrating its effectiveness and potency. Our new article improves this study by utilizing a hybrid method to enhance the deduplication process speed. Additionally, we enhance the deduplication ratio by incorporating triple divisors into our approach.

S. Babu, P. Ramya, and J. Gracewell,2022 [24] introduced a content deduplication with granularity tweak (CDGT) technique within the Hadoop architecture, specifically targeting large text datasets. CDGT utilizes the Reed-Solomon technique to enhance deduplication efficiency by effectively handling small changes within similar content. It achieves this by verifying intercontact changes and identifying and eliminating a more significant amount of duplicate content. An indexing approach improves performance by organizing data into clusters, facilitating faster access and retrieval operations. However, one drawback of this technique is its limited effectiveness in handling large text datasets. To address this limitation, our proposed method is designed to apply to different data types. To validate the performance and suitability of our approach, we conducted comprehensive experiments using multiple datasets, ensuring a thorough evaluation across various data types.

Y. Deng, 2022 [25] The proposed technique improves deduplication-based backup systems by addressing the challenges of index lookup bottleneck and data fragmentation. It introduces HID (hot fingerprint entry distilling) to segregate useless and fragmented fingerprint entries, optimizing memory utilization and reducing disk accesses. EHID (Evolved HID) incorporates a Bloom filter to identify unique and fragmented chunks, avoiding disk accesses and reducing false positives. Experimental results show significant reductions in memory overhead, improved backup throughput, and reduced disk I/O traffic compared to the state-of-the-art method HAR. The main drawback of this method of dividing the index into three parts (hot, fragmented, and useless fingerprint entries) and using containers based on a threshold introduces complexity to the deduplication process. This additional layer of management and classification increases the implementation and maintenance effort needed for the deduplication system. Therefore, we improve the index lookup by retrieving chunks of the same size and divisors; after that, we compute the three hash for these chunks and compare them with the new chunks.

3. Proposal system

This paper introduces a hybrid data deduplication system combining two techniques. The first technique is file-level deduplication, which treats each file as a single entity and does not divide it into smaller units. The second technique is chunk-level deduplication, which divides data into various chunks of different sizes. The proposed system consists of three phases: file-level Deduplication, block or chunk-level Deduplication, and data indexing and storage. Fig. 1 illustrates the three main phases of the proposed system.

3.1 Multi-hash function algorithm

A new hash function is introduced in the proposed system to generate the fingerprint of the input file or chunk. This function computes a multi-hash function for each file or chunk, reducing the computation, storage space, and collision issues that conventional deduplication system hash functions face. The

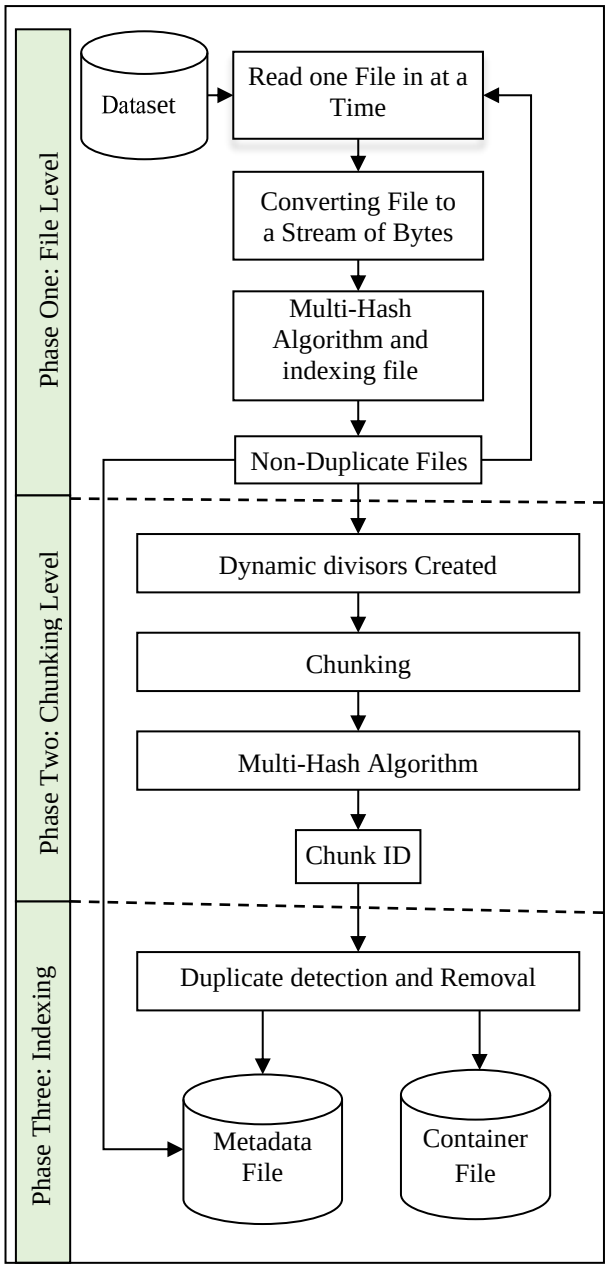

Figure. 1 The layout of the proposed system

proposed multi-hash algorithm employs a fundamental mathematical concept to reduce processing time and system resources.

Algorithm 1 part (1) is designed to generate three random sequences of hash arrays for a given file. The algorithm takes a file as a stream of bytes as input and produces three output arrays: h1, h2, and h3, each containing 255 random values. The algorithm generates three random sequences of hash arrays by shuffling a sequence array using a random number generator. The shuffling process ensures the randomness of the generated sequences. These sequences can be used for various purposes, such as fingerprint operations.

Algorithm 1: Part 1: Create three random hash values for each file

Objective:	Generate three random sequences of hash arrays.
Input:	File as a stream of byte
Output:	h1, h2, and h3: three sequence arrays, each containing 255 random values.
Step 1:	Define the following integer vector arrays: h1[255], h2[255], h3[255], seq[255]
Step 2:	Initialization: Fill the seq array with sequence values from 1 to 255.
Step 3:	Compute the length of the stream of bytes and assign it to L.
Step 4:	Loop from i = 254 to 0: - Generate a random value j between 0 and i (inclusive). - Swap the values of seq[i] and seq[j]. - Set seq[i] as L. - Set seq[j] as L. Move to the next iteration.
Step 5:	Loop from L = 0 to 254: - Assign the value of seq[L] to h1[L]. - Assign the value of seq[L] to h2[L]. - Assign the value of seq[L] to h3[L]. Move to the next iteration.
Step 6:	Return the three random sequence arrays: h1, h2, and h3.

As demonstrated in the Algorithm 1 part 2, the hashing algorithm generates hash values for each file or chunk. It focuses on generating three hash functions for a given file. The algorithm's input is the file as a stream of bytes and three sequence arrays (h1, h2, and h3). It produces three output arrays: hash1, hash2, and hash3, representing the three hash functions. Three hash values are produced by iterating through a file's byte stream and performing multiplication and addition operations using sequence arrays.

The proposed hashing method calculates three values for every file or chunk, with each hash occupying two bytes. Consequently, each file requires 6 bytes (48 bits) in total. Traditional hash algorithms such as SHA1, SHA256, and MD5, commonly used for data deduplication, demand extra system resources to generate hash values and store them on storage media. For example, SHA1 necessitates 160 bits of data, while MD5 requires 128 bits. Eq (1), Eq (2), and Eq (3) show how

Algorithm 1: part2: Generate Three Hash Functions for Each File

Objective: Calculate three hash values.	
Input:	File as a stream of bytes, h1, h2, and h3 (three sequence arrays generated from Algorithm 1, Part 1).
Output:	hash1, hash2, and hash3 (three hash functions).
Step1:	Define the following integer vector arrays: hash1[], hash2[], hash3[]
Step2:	Initialization: Fill hash1, hash2, and hash3 arrays with 0 values.
Step3:	Assign the byte stream of the file to the variable s.
Step4:	Loop from j = 1 to the end of the file: - Calculate k = j mod 255. - Update hash1 as (hash1 + s[j] × h1[k]) modulo 65535. - Update hash2 as (hash2 + s[j] × h2[k]) modulo 65535. - Update hash3 as (hash3 + s[j] × h3[k]) modulo 65535. Move to the next iteration.
Step5:	Return the three hash functions: hash1, hash2, and hash3.

mathematically it generates three hash functions.

$$Hash1 = \sum_{j=1}^{n}\left([s[j] \times h1[j \bmod 255]]\right) mod\ 65536 \quad (1)$$

$$Hash2 = \sum_{j=1}^{n}([s[j] \times h2[j \bmod 255]])\ mod\ 65536 \quad (2)$$

$$Hash3 = \sum_{j=1}^{n}\left([s[j] \times h3[j \bmod 255]]\right) mod\ 65536 \quad (3)$$

In these equations: n represents the length of the file and the loop iterates from j = 1 to n.

These equations calculate the updated values of hash1, hash2, and hash3 by summing the product of each byte s[j] with the corresponding element h1[j mod 255], h2[j mod 255], or h3[j mod 255] from the sequence arrays. The modulo operation is applied to ensure the result stays within the range of 0 to 65534.

3.2 Files level deduplication phase

The proposed hybrid system utilizes file-level Deduplication to remove duplicate files by considering each file as a single entity and entering it into the deduplication system as a stream of bytes.

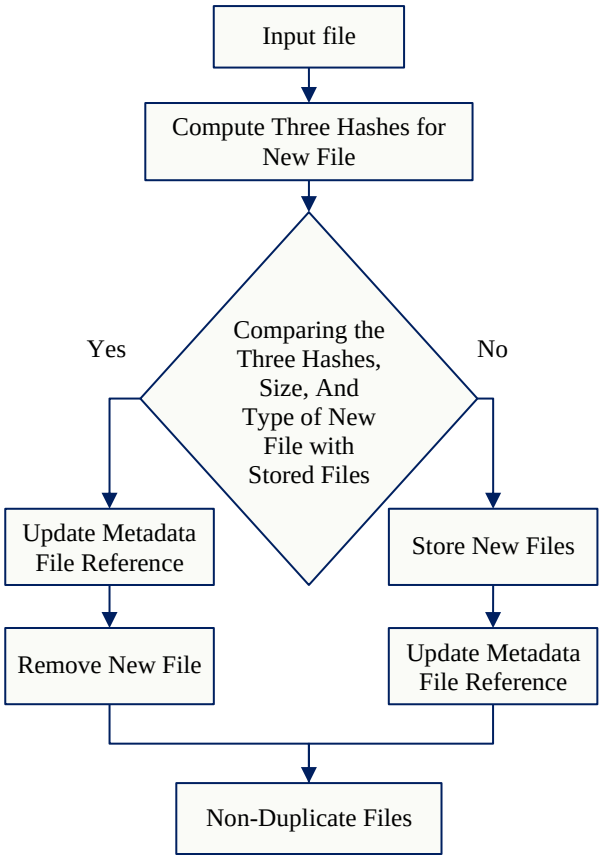

Figure. 2 The File Level deduplication

Fig. 2 depicts the critical stages of the file-level Deduplication technique. Using file-level deduplication eliminates feeding duplicated files into the second phase of the deduplication system. It provides several benefits, including creating only one index per file, saving time and space, reducing the number of index values stored, and minimizing CPU usage and I/O operations.

However, file-level deduplication may not detect changes made to a small portion of the file. The proposed system uses a hybrid approach to overcome the limitation of file-level deduplication. If changes occur in a file's bytes, the system resolves the issue in the second phase by breaking the file into chunks using variable-sized chunk-level deduplication.

3.2.1. File identification and comparison

A unique identifier (ID) is generated for each file to enable efficient file identification and comparison. This ID comprises the file's size, type (extension), and three hash functions. Files are classified based on size and type (extension) to expedite comparison, resulting in several groups of similar files. When a new file is introduced, it is only compared with files in the group that share the same size and type. This grouping method accelerates comparison operations and facilitates the detection of identical files. For

example, suppose the new file matches previous files with identical size, type (extension), and fingerprint (three hashes); in that case, it is eliminated and replaced by a logical reference pointing to the dataset's current file. In contrast, if the new file is unique, its bytes are stored on the disk.

3.3 Chunking level deduplication phase

During the chunking level phase, the non-duplicate files obtained from the file level phase are segmented into several chunks. This paper describes a novel method to improve the TTTD algorithm by dividing files into several chunks using a dynamic list of divisors. The list of divisors is constructed depending on the content and size of the dataset, and it defines the file chunking breakpoints.

3.3.1. Dynamic list of divisors

A new method is proposed to generate a list of divisors that leverage statistical analysis of a dataset's content. The proposed system utilizes statistical analysis to determine each contiguous triple-byte frequency, which is then utilized as divisors to partition files. The divisors are automatically adapted according to the size and nature of the file to enhance the partitioning process. The resulting divisors are then employed as division points to break the data stream into smaller, more easily manageable parts. The new technique is practical and feasible for generating divisors to partition datasets into manageable segments. The proposed system follows a three-step process to find the divisors:

Step 1: computing the frequency of each adjacent triple byte, i.e., the number of times it appears in a dataset, resulting in a set of divisors based on the triple byte frequency.

Step 2: The list is sorted in descending order based on the frequency of the triple bytes.

Step 3: Selects a set number of triple bytes with the highest frequency as divisors for each group.

The optimal number is determined through experimentation to achieve the best deduplication results.

3.3.2. File chunking

This article introduces a new chunking technique that enhances the efficiency of the TTTD algorithm. As shown in Fig. 3, The new technique determines the breakpoints based on the most frequent triple byte in the dataset. The technique incorporates a minimum chunk size (Tmin) to prevent the generation of small chunks and a maximum chunk size (Tmax) to avoid forming large chunks. The breakpoints are established based on the list of divisors with high frequency, and the scanning process begins at Tmin and continues until Tmax to detect the breakpoints. If no divisors are found within this range, Tmax is used as the breakpoint. The last breakpoint to the end of the file may form a (Tail) greater than 0 bytes but less than Tmax, and no is found. After determining the breakpoints based on the most frequent triple byte in the dataset, the files are divided into chunks of varying sizes.

3.3.3. Chunks hashing

Multi-hash function: The proposed system introduces a new hashing method that computes multi-hash values for each chunk, as described in the Algorithm (1) parts 1 and 2. where 48 bits are required to hash each chunk of data, providing an efficient and cost-effective solution. Hash functions are critical for efficiently detecting duplicate data, as comparing bytes to identify duplicates can be time-consuming and require multiple input/output operations. Data fingerprinting is a preferred approach for identifying duplicates, and hash functions are crucial. If two data chunks are identical, the same data's hash functions are also identical.

3.4 Indexing and comparison phase

The third phase of the proposed system's is lookup and comparison. For each chunk, the metadata file stores the chunk size, divisors, and position (induct to position chunk in the container file). The non-duplicated chunk's value is stored in a container file that includes the chunk value and position. As illustrated in Fig. 4, this paper introduces a new method involving grouping chunks into multiple categories, significantly reducing search spaces and accelerating the matching process. The proposed method involves a multilevel hierarchical search and matching mechanism to detect duplicate chunks when new chunks are added. The lookup and Matching procedure can be summarized as follows:

- The new file is divided into chunks using the proposed algorithm that depends on the list of divisors of the file's contents.
- An identifier ID is calculated for each chunk, consisting of (chunk size, divisors, hash1, hash2, and hash3).
- A set of records with the exact (chunk size and divisors) is retrieved from the metadata.

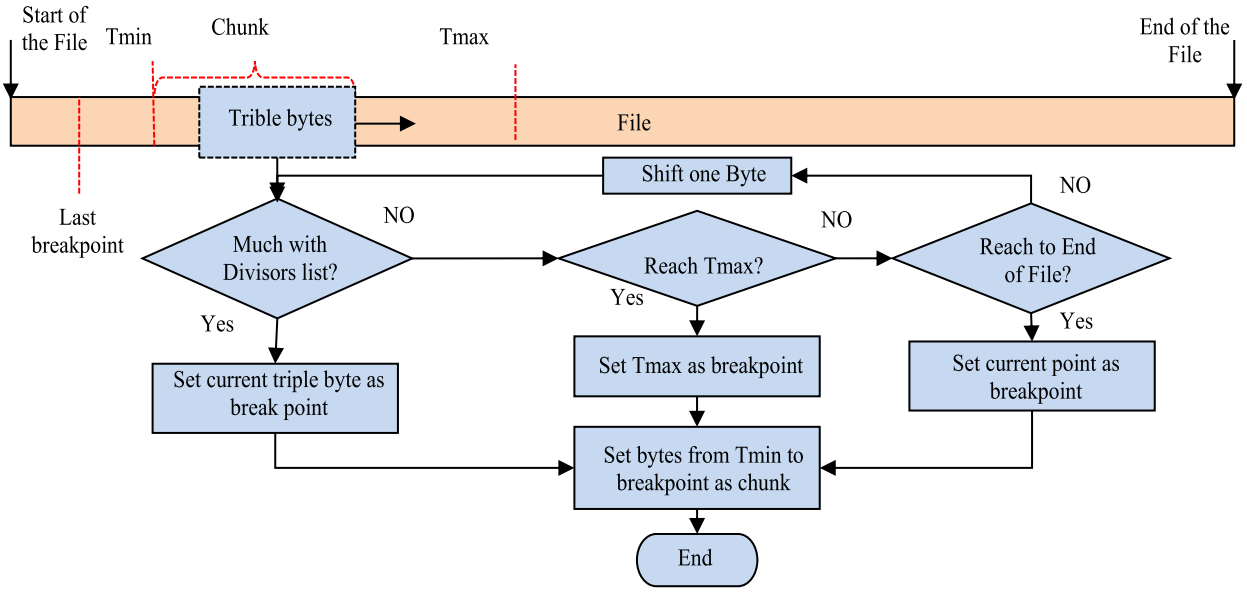

Figure. 3 Chunking file using proposal triple list divisors

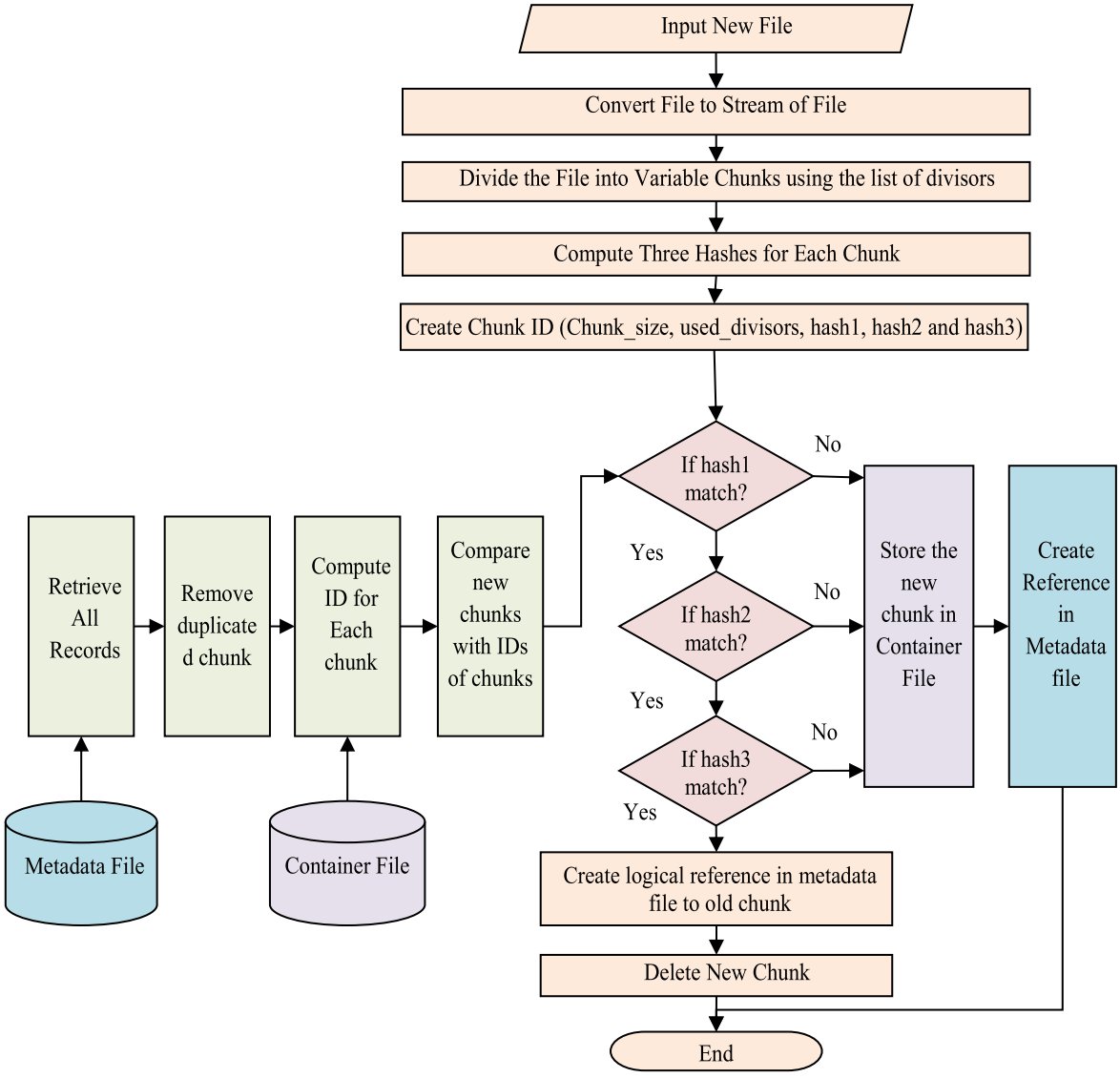

Figure. 4 Lookup and Incremental Matching

- Three hashes are counted for each chunk retrieved from the backup storage, creating a Chunk ID.
- If the chunk size, used divisors, and the first hash of the two chunks (new chunk and old chunks) are the same, then the second hash of the two chunks is compared, followed by the third.

When there is a match between the two chunks, update the existing chunk's reference where a new reference is added to the metadata file, indicating the existing chunk and deleting the new duplicated chunk. If the new chunk is non-duplicated, it is added to the container file, and its reference is saved in the metadata file.

3.5 Dataset and computer description

The system was evaluated on an Intel (R) Core (TM) i5-10300H CPU with four cores, 16.0 GB RAM, running Windows 11 operating system, and programmed using C# Visual Studio 2022. seven datasets containing files of varying sizes, types, and properties were used to test the system's performance:

1. Dataset 1: SQLite Sources, comprising 190,703 files with a combined size of 6.50 gigabytes (GB).
2. Dataset 2: three-dimensional drawings plus the initials of the author Laurence D. Finston (3DLDF) files of (GNU's Not Unix) GNU source code versions comprising 5,795 files with a data size of 2.27 GB.
3. Dataset 3: Linux source code, consisting of 496,867 files with a total size of 6.58 GB.
4. Dataset 4: The DUC2004 dataset is a specialized collection for evaluating document summarization techniques. It comprises 500 news articles, each accompanied by four human-written summaries. The dataset is organized into 50 clusters of Text REtrieval Conference (TREC) documents.
5. Dataset5: The TAR dataset consists of 95 compressed files containing source code from open-source projects like MySQL, GCC, and Glibc. The total size of the dataset is approximately 56GB.
6. Dataset6: consists of multiple versions of the Linux source code, encompassing different releases and revisions of the operating system. The dataset's total size is 99GB, indicating the cumulative space required to store these various versions of the Linux source code.
7. Dataset7: comprises three versions of the Linux file system, namely Linux3.9, Linux-4.14.157, and Linux-5.8.12. This dataset encompasses 173,109 files and 11,705 folders, approximately 2.32 GB.

4. Experiments and results

The proposed methods were evaluated based on four criteria:

Deduplication Time: refers to the time required for the deduplication technique to provide an output response.

Duplicate elimination ratio (DER*):* is a comprehensive deduplication metric computed by dividing the input data size by the output data size, as presented in Eq. (4). This ratio indicates the effectiveness of the deduplication system [10].

$$\text{DER} = \frac{\text{Data Size Before Deduplication}}{\text{Data Size After Deduplication}} \tag{4}$$

Throughput: It is the rate at which data is processed and deduplicated in a given period, and it is commonly expressed in units such as bits per second (bps), megabits per second (Mbps), or gigabits per second (Gbps). It is calculated as in Eq. (5) [10].

$$\text{Throughput} = \frac{\text{Processed Data}}{\text{Time in Second}} \tag{5}$$

Deduplication Gain: It is a metric that measures the reduction in storage space or processing resources achieved by removing duplicate data. It is calculated as in Eq. (6) [24].

$$\text{Gain} = 1 - \frac{Datasize\ after\ deduplication}{Datasize\ before\ deduplication} \tag{6}$$

The proposed hybrid method achieved the best throughput results across all three datasets. Fig. 5 illustrates the throughput results of non-hybrid deduplication and the proposed hybrid Deduplication system. Note that the proposed hybrid system's throughput is higher than the non-hybrid system.

Table 1 compares the time required for non-hybrid and hybrid Deduplication systems for three datasets. The hybrid deduplication consistently required less time than the non-hybrid deduplication,

Table 1. Compares the time of the proposed hybrid deduplication and non-hybrid deduplication system

Dataset	Non-Hybrid Time (Sec)	Hybrid (Sec) Time (Sec)
Dataset1	2143	180
Dataset2	232	156.6
Dataset3	7446	952

Table 2. Time and DER of file-level and block-level and the hybrid between them

Dataset	File-level		Block level		Hybrid	
	Time (Sec)	DER	Time (Sec)	DER	Time (Sec)	DER
Dataset1	45	7.7	135	29.0	180	36.7
Dataset2	12	2.8	144	9.3	156.6	12.1
Dataset3	180	2.9	772	2.7	952	5.61

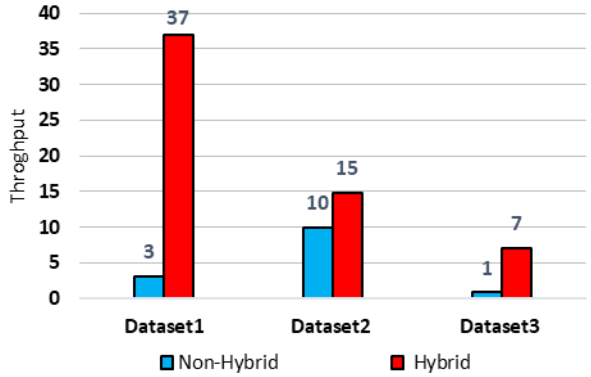

Figure. 5 A comparison of the throughput of the non-hybrid and the hybrid deduplication

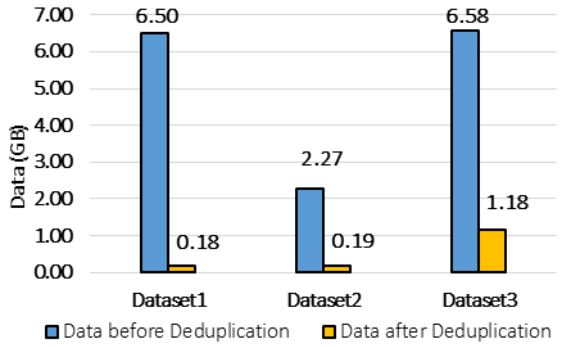

Figure. 6 Data size before and after the hybrid deduplication method

indicating its effectiveness in reducing the time required for data deduplication.

Table 2 presents the data deduplication ratio and time required for the file-level and block-level deduplication methods and the hybrid between them. It provides data for three datasets.

Fig. 6 displays the data size before and after the proposed hybrid deduplication system. The analysis indicates a substantial reduction in the amount of data, with Dataset 1 experiencing a reduction of 97.2%, from 6.50 GB to 0.18 GB. Dataset 2 had a reduction of 91.6%, while Dataset 3 had a reduction of 82.1%. These outcomes demonstrate the effectiveness of the proposed hybrid method in reducing the data size.

Table 3 displays the results of experimentation that determined the optimal number of divisors

Table 3. DER and time consumption based on the number of divisors (1-10)

Divisors	Dataset1		Dataset2	
	DER	Time (Sec)	DER	Time (Sec)
1	32.1	1600	10.88	150
2	33.9	1935	11.28	153
3	36.1	2025	11.27	155
4	36.7	2143	12.08	156.6
5	36.4	2166	12.15	156.9
6	35.5	1943	12.07	190
7	35.0	1996	12.12	201
8	35.0	2019	12.15	199
9	34.8	2385	12.15	210
10	34.6	2553	12.17	232

Table 4. Compared the proposed hashing method with MD5, SHA-1 and SHA256 by considering time criteria

Dataset	Hashing Time (Sec)			
	MD5	SHA1	SHA256	Proposed Hashing
Dataset1	50	192	576	25.9
Dataset2	24.8	60.1	185	7.3
Dataset3	424.2	462.9	1320	67

required to achieve the best deduplication outcomes. Datasets 1 and 2 displayed the highest DER when 4 and 5 divisors were utilized, respectively. Nonetheless, employing a single divisor for both datasets resulted in the least time-consuming approach, despite the low DER outcome.

Furthermore, Table 3 illustrates a direct relationship between the number of divisors and the time required, as an increase in divisors corresponds to an increase in time consumption.

The study demonstrates that the proposed hashing technique surpasses SHA1, SHA256 and MD5 in terms of performance, resulting in reduced hashing time, as evidenced in Table 4. These results substantiate the effectiveness of the proposed technique in improving the efficiency of the data deduplication system.

Table 5 showcases how the proposed hashing technique outperforms SHA1, SHA256 and MD5, leading to a notable increase in the data deduplication system's throughput. These findings strongly endorse the proposed technique's efficacy in enhancing the system's overall efficiency.

Table 6 presents a comprehensive comparison of the proposed data deduplication method with various state-of-the-art methods, focusing on the gain percentage and deduplication elimination ratio

Table 5. Compared the proposed hashing method with MD5, SHA1 and SHA256 by considering throughput

Dataset	Hashing Throughput (Mb/s)			
	MD5	SHA1	SHA256	Proposed Hashing
Dataset1	133	34.66	11.5	257
Dataset2	93.67	38.28	12	318
Dataset3	13.4	12.4	5.1	100

Table 6. Compared the proposed method with other state-of-the-art methods

Ref.	Methods	Dataset	Gain	DER
[1]	CB-TTTD	Dataset2	91%	11.6
[10]	BFBC	Dataset1	96%	30.82
		Dataset3	80%	4.96
[22]	CRP	Dataset7	68%	3.11
[16]	QuickCDC	Dataste5	41%	1.7
[23]	File classification based on histogram	Dataset1	96%	27.17
		Dataset2	91%	11.65
[24]	CDGT	Dataste4	84%	7.3
[25]	EHID	Dataset6	96.7%	33
Ours	Proposed method	Dataset1	97.2%	36.7
		Dataset2	91.6%	12.1
		Dataset3	82.1%.	5.61
		Dataset7	71%	3.5

(DER) achieved by each method. The findings highlight the superior performance of the proposed method compared to others, primarily attributed to its ability to achieve significantly high deduplication gain and DER, thereby establishing itself as the most efficient approach.

For Dataset1, the proposed method achieves an impressive gain percentage of 97.2% alongside a DER of 36.7. Similarly, for Dataset2, it attains a gain of 91.6% with a DER of 12.1. In the case of Dataset3, the method achieves a commendable gain of 82.1% and a deduplication ratio of 5.6. In general, these results clearly indicate that the proposed method outperforms the majority of other approaches in terms of gain percentage and DER.

5. Conclusion

This paper presents a new hybrid system that effectively combines file- and chunk-level deduplication approaches to reduce data redundancy. The method's effectiveness is evaluated using three datasets of varied sizes and kinds, measuring deduplication time, DER, and throughput between the hybrid and non-hybrid approaches. The results indicate that the hybrid approach saves significant time compared to the non-hybrid method across all three datasets, with reductions of 97.2%, 91.6%, and 82.1% in data size for Dataset 1, Dataset 2, and Dataset 3, respectively. By examining file contents and producing a list of divisors, the hybrid approach achieves the maximum Throughput and DER. The suggested hash algorithm outperforms SHA1, SHA256 and MD5 regarding hash throughput and time. Additionally, the proposed approach is compared to other state-of-the-art methods, such as CB-TTTD, BFBC, CRP, QuickCDC, CDGT and EHID, and is shown to be superior in terms of deduplication ratio, deduplication gain.

Conflicts of interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Author contributions

In this research article, the author's contributions are as follows: "conceptualization, Loay E. George; methodology, Loay E. George; software, Hashem B. Jehlol; validation, Hashem B. Jehlol; formal analysis, Loay E. George; investigation, Hashem B. Jehlol; resources, Hashem B. Jehlol; data curation, Loay E. George; writing— original draft preparation, Hashem B. Jehlol; writing—review and editing, Hashem B. Jehlol; visualization, Hashem B. Jehlol; supervision, Loay E. George; project administration, Loay E. George; funding acquisition, Hashem B. Jehlol."

References

[1] H. Jasim and A. Fahad, “New techniques to enhance data deduplication using content based-TTTD chunking algorithm”, *Int. J. Adv. Comput. Sci. Appl.*, Vol. 9, No. 5, pp. 116–121, 2018.

[2] S. Kareem, R. Yousif, and S. Abdalwahid, “An approach for enhancing data confidentiality in hadoop”, *Indones. J. Electr. Eng. Comput. Sci.*, Vol. 20, No. 3, pp. 1547–1555, 2020.

[3] S. Chimphlee and W. Chimphlee, “Machine learning to improve the performance of anomaly-based network intrusion detection in big data”, *Indones. J. Electr. Eng. Comput. Sci.*, Vol. 30, No. 2, pp. 1106–1119, 2023.

[4] Y. Cui, Z. Lai, X. Wang, and N. Dai, “QuickSync: Improving Synchronization Efficiency for Mobile Cloud Storage Services”, *IEEE Trans. Mob. Comput.*, Vol. 16, No. 12, pp. 3513–3526, 2017.

[5] F. Ni, X. Lin, and S. Jiang, “SS-CDC: A two-stage parallel content-defined chunking for

deduplicating backup storage", *SYSTOR - Proc. 12th ACM Int. Syst. Storage Conf.*, pp. 86–96, 2019.

[6] H. Fan, G. Xu, and Y. Zhang, "CSF: An efficient parallel deduplication algorithm by clustering scattered fingerprints", In: *Proc. IEEE Intl Conf Parallel Distrib. Process. with Appl. Big Data Cloud Comput. Sustain. Comput. Commun. Soc. Comput.*, pp. 602–607, 2019.

[7] S. Mohamed and Y. Wang, "A survey on novel classification of deduplication storage systems", *Distrib. Parallel Databases*, Vol. 39, No. 1, pp. 201–230, 2021.

[8] V. Devarajan and R. Subramanian, "Analyzing semantic similarity amongst textual documents to suggest near duplicates", *Indones. J. Electr. Eng. Comput. Sci.*, Vol. 25, No. 3, pp. 1703–1711, 2022.

[9] D. Viji and D. Revathy, "Comparative Analysis for Content Defined Chunking Algorithms in Data Deduplication", *Webology*, Vol. 18, No. Special Issue 2, pp. 255–268, 2021.

[10] A. Saeed and L. E. George, "Data deduplication system based on content-defined chunking using bytes pair frequency occurrence", *Symmetry*, Vol. 12, No. 11, pp. 1–21, 2020.

[11] E. Manogar, "A smart hybrid content de ned chunking algorithm for data deduplication in cloud storage", *Anna Univ. Chennai Abirami*, 2022.

[12] Y. Zhang, Y. Wu, and G. Yang, "Droplet: A distributed solution of data deduplication", In: *Proc. of IEEE/ACM Int. Work. Grid Comput.*, pp. 114–121, 2012.

[13] L. Canencia and B. Hamoum, "Deduplication algorithms and models for efficient data storage", In: *Proc. of 24th Int. Conf. Circuits, Syst. Commun. Comput. CSCC 2020*, pp. 23–28, 2020.

[14] C. Zhang, "MII: A novel content defined chunking algorithm for finding incremental data in data synchronization", *IEEE Access*, Vol. 7, No. 1, pp. 86932–86945, 2019.

[15] W. Xia, D. Feng, H. Jiang, Y. Zhang, V. Chang, and X. Zou, "Accelerating content-defined-chunking based data deduplication by exploiting parallelism", *Futur. Gener. Comput. Syst.*, Vol. 98, No. January 2021, pp. 406–418, 2019.

[16] Z. Xu and W. Zhang, "QuickCDC: A Quick Content Defined Chunking Algorithm Based on Jumping and Dynamically Adjusting Mask Bits", In: *Proc. of 19th IEEE Int. Symp. Parallel Distrib. Process. with Appl.*, pp. 288–299, 2021.

[17] H. Abdulsalam and A. Fahad, "Evaluation of Two Thresholds Two Divisor Chunking Algorithm Using Rabin Finger print, Adler, and SHA1 Hashing Algorithms", *Iraqi J. Sci.*, Vol. 58, No. 4C, 2017.

[18] K. Eshghi and H. Tang, "A framework for analyzing and improving content-based chunking algorithms", *Hewlett-Packard Labs Tech. Rep. TR*, No. August, 2005, [Online]. Available: http://shiftleft.com.

[19] L. George and A. Saeed, "Data Deduplication System Based on Frequency Occurrence", *Symmetry (Basel).*, Vol. 12, No. 11, p. 1841, 2020.

[20] A. Saeed and L. George, "Fingerprint-based data deduplication using a mathematical bounded linear hash function", *Symmetry (Basel).*, Vol. 13, No. 11, pp. 1–19, 2021.

[21] A. Bhalerao, "A Survey : On Data Deduplication for Efficiently Utilizing Cloud Storage for Big Data Backups", *Int. Conf. Trends Electron. Informatics*, No. August 2019, 2017.

[22] S. Ahmed and L. George, "Lightweight hash-based de-duplication system using the self detection of most repeated patterns as chunks divisors", *J. King Saud Univ. - Comput. Inf. Sci.*, Vol. 34, No. 7, pp. 4669–4678, 2021.

[23] H. Jehlol and L. George, "Big Data De-duplication Using Classification Scheme based on Histogram of File Stream", In: *Proc. of International Conference on Intelligent Technology, System and Service for Internet of Everything*, 2022.

[24] S. Babu, P. Ramya, and J. Gracewell, "Content Deduplication with Granularity Tweak Based on Base and Deviation for Large Text Dataset", *Hindawi Scientific Programming*, pp. 1-17, 2022.

[25] Z. Datong, D. Yuhui, and Z. Yi , "Improving the Performance of Deduplication-Based Backup Systems via Container Utilization Based Hot Fingerprint Entry Distilling", *ACM Transactions on Storage*, Vol. 17, No. 4, pp. 1–23, 2021.