symmetry

Article

Data Deduplication System Based on Content-Defined Chunking Using Bytes Pair Frequency Occurrence

Ahmed Sardar M. Saeed [1,*] and Loay E. George [2]

1 Information Technology, Technical College of Informatics, Sulaimani Polytechnic University, Sulaymanyah 46001, Iraq
2 Assistant of University President for Scientific Affairs, University of Information Technology and Communication (UoITC), Baghdad 10011, Iraq; loayedwar57@scbaghdad.edu.iq
* Correspondence: ahmed.sardar@spu.edu.iq; Tel.: +964-770-110-5766

Received: 13 September 2020; Accepted: 4 November 2020; Published: 6 November 2020

Abstract: Every second, millions of data are being generated due to the use of emerging technologies. It is very challenging to store and handle such a large amount of data. Data deduplication is a solution for this problem. It is a new technique that eliminates duplicate data and stores only a single copy of data, reducing storage utilization and the cost of maintaining redundant data. Content-defined chunking (CDC) has been playing an important role in data deduplication systems due to its ability to detect high redundancy. In this paper, we focused on deduplication system optimization by tuning relevant factors in CDC to identify chunk cut-points and introduce an efficient fingerprint using a new hash function. We proposed a novel bytes frequency-based chunking (BFBC) algorithm and a new low-cost hashing function. To evaluate the efficiency of the proposed system, extensive experiments were done using two different datasets. In all experiments, the proposed system persistently outperformed the common CDC algorithms, achieving a better storage gain ratio and enhancing both chunking and hashing throughput. Practically, our experiments show that BFBC is 10 times faster than basic sliding window (BSW) and approximately three times faster than two thresholds two divisors (TTTD). The proposed triple hash function algorithm is five times faster than SHA1 and MD5 and achieves a better deduplication elimination ratio (DER) than other CDC algorithms. The symmetry of our work is based on the balance between the proposed system performance parameters and its reflection on the system efficiency compared to other deduplication systems.

Keywords: data deduplication; content-defined chunking; bytes frequency-based chunking; data deduplication gain; hashing; deduplication elimination ratio

1. Introduction

The amount of digital data is rising explosively, and the forecasted amount of data to be generated by the end of 2020 is about 44 zettabytes. Because of this "data flood," storing and maintaining backups for such data efficiently and cost-effectively has become one of the most challenging and essential tasks in the big data domain [1–3]. Enterprises, IT companies, and industries need to store and operate on an enormous amount of data. The big issue is how to manage these data. To manage data in a proper way, data deduplication techniques are used. Dropbox, Wuala, Mozy, and Google Drive employ deduplication techniques to reduce cloud storage capacity and utilize cloud storage more appropriately. Data deduplication is becoming a dominant technology to reduce the space requirement for both primary file systems and data backups [4,5]. It is an effective data reduction method that not only reduces storage space by removing redundant data and ensuring that there is only one single copy of data present in storage but also minimizes the transmission of redundant data in low-bandwidth network environments. In addition, it helps to reduce the cost associated with large scale data backups [1,6]. Data deduplication can work at the file or block level. In file-level deduplication, if two files are completely the same, one copy of the file is stored. In block-level deduplication, the software looks inside a file and saves the unique data of each block. If a file is modified, only the changed data are stored. This is a far more efficient procedure than file-level deduplication [7]. Generally, the four major steps involved in most block-level data deduplication approaches are chunking, fingerprinting, indexing of fingerprints, and storage management [3,8,9].

A typical block-level data deduplication system divides the input data stream into multiple data "chunks" that are each individually identified and duplicate-detected by a cryptographic secure hash signature (e.g., MD5 and SHA1), also known as a fingerprint. These chunks can be fixed- or variable-sized units, determined by the content itself, through a process called content-defined chunking (CDC) [10]. According to recent studies, variable-size CDC techniques detect more redundancy than fixed-size chunking, and avoid the boundary-shift problem during the insertion or deletion of data [2,11,12].

The CDC algorithm has a significant effect on the deduplication ratio and performance, as the chunking stage has a direct impact on finding duplications. The CDC sliding-window-based algorithm is not optimal, as the output window must be slid according to each byte. Based on the predefined condition, a special judgement function is computed at each byte to decide whether the content window satisfies that condition. The two thresholds two divisors (TTTD) algorithm introduces another chunking condition to enhance the deduplication ratio, but it does not enhance the deduplication performance due to the highly CPU-intensive results from the chunking and fingerprinting steps. Most of the research focuses on enhancements in the fingerprint indexing and matching stages, while the chunking and chunk fingerprinting steps still need more attention [13].

In this paper, a new CDC algorithm with a novel procedure is proposed. For efficient chunking, we propose the bytes frequency-based chunking (BFBC) technique as a fast and efficient chunking algorithm, where we expressively reduce the number of computing operations by using multi dynamic optimum parameter divisors (D) with the best threshold value, exploit the multi-operational nature of BFBC to reduce the chunk-size variance, and maximize chunking throughput with an improved deduplication elimination ratio (DER). In addition, a new hashing function with a straightforward judgment function is suggested to improve the matching process and decrease the probability of hash collision occurrence. Reducing the computational complexity of chunking and chunk fingerprinting steps will make room for CPU resources to perform other tasks.

The contributions of this work are as follows:

1. The proposed system defines chunk boundaries based on the bytes frequency of occurrence instead of the byte offset (as in the fixed-size chunking technique), so any change in one chunk will not affect the next one, and the effect will be limited to the changed chunk only.
2. Content-defined chunking consumes substantial processing resources to calculate hash values using SHA1 or MD5 for data fingerprinting, while the proposed system uses mathematical functions to generate three hashes that consume fewer computing resources. Furthermore, compared to the traditional TTTD method, the number of bits needed to store these three hashes is 48 bits, which is less than the number of bits needed to save the hash value in SHA1 (160 bits) and MD5 (128 bits).

The details of the proposed system will be presented in the remainder of the paper. Section 2 illustrates works related to data deduplication. In Section 3, system methodology is discussed. In Section 4, the proposed system is described. In Section 5, the results of our suggested method are discussed. The last section provides conclusions and addresses future works.

2. Related Work

Data deduplication systems have been subject to intensive research for the last few years. They generally detect redundant objects by comparing calculated fingerprints rather than comparing byte by byte and transparently eliminating them [14]. At present, researchers mainly focus on improving data chunking algorithms to increase the deduplication elimination ratio (DER). Bin Zhou et al. [15] explained that most current chunking algorithms use the Rabin algorithm sliding window for the chunking stage, which utilizes a large amount of CPU resources and leads to performance issues. Accordingly, they proposed a new chunking algorithm, namely, the bit string content aware chunking strategy (BCCS), which calculates the chunk's fingerprint using a simple shift operation to reduce the resource utilization. Venish and Sankar [10] discussed different chunking methods and algorithms and assessed their performances. They found that an effective and efficient chunking algorithm is crucial. If the data are chunked precisely, it increases the throughput and the deduplication performance. Compared to file-level chunking and fixed-size chunking, the content-based chunking approaches deliver good throughput and reduce space consumption.

Wang et al. [16] presented a logistically based mathematical model to enhance the DER based on the expected chunk size, as the previously proposed 4 KB or 8 KB chunk size did not provide the best optimization for the DER. To validate the correctness of the model, they used two realistic datasets, and according to the results, the R2 value was above 0.9. Kaur et al. [17] presented a comprehensive literature review of existing data deduplication techniques and the various existing classifications of deduplication techniques that have been based on cloud data storage. They also explored deduplication techniques based on text and multimedia data along with their corresponding classifications, as these techniques have many challenges for data duplication detection. They also discussed existing challenges and significant future research directions in deduplication. Zhang et al. [18] proposed a new CDC algorithm called the asymmetric extremum (AE), which has higher chunking throughput and smaller chunk size variance than the existing CDC algorithms and an improved ability to chunk boundaries in low-entropy strings. The system shows an enhancement in performance and reduces the bottleneck of chunking throughput while maintaining deduplication efficiency. Nie et al. [19] developed a method to optimize the deduplication performance by analyzing the chunking block size to prevent blocks that are too large or too small, which affects the data deduplication efficiency. It was proved that selecting the optimum block size for the chunking stage will improve the data deduplication ratio. Fan Ni [20] quantified the impact of the existing parallel CDC methods on the deduplication ratio, and proposed a two-phase CDC method (SS-CDC) that can provide substantially increased chunking speed, as can regular, parallel CDC approaches, and achieve the same deduplication ratio as the sequential CDC method does. An opportunity was identified in a journaling file system where fast non-collision-resistant hash functions can be used to generate weak fingerprints for detecting duplicates, thus avoiding the write-twice issue for the data journaling mode without compromising data correctness and reliability. Xia et al. [21] proposed a fast CDC approach for data deduplication. The main idea behind it is the use of five key techniques, namely, a gear-based fast rolling hash, optimizing the gear hash judgment for chunking, subminimum chunk cut-point skipping, normalized chunking, and two-byte rolling. The experimental results proved that the proposed approach gains a chunking speed that is about 3–12× higher than the state-of-the-art CDC, while nearly accomplishing the same or a higher deduplication ratio with respect to the Rabin-based CDC. Taghizadeh et al. [22] developed an intelligent approach for data deduplication on flash memories. Based on data content and type, there was a classification for the write requests, and the metadata for it was stored as separate categories to enhance the search operation, resulting in an improvement in the search delay and enhancing the deduplication rate considerably.

The core enabling technologies of data deduplication are chunking and hashing. The performance bottleneck caused by the computation-intensive chunking and hashing stages of data deduplication presents a significant challenge. Most of the above research worked on optimizing these two stages, as minimizing the chunking and hashing overhead are becoming increasingly urgent aims for deduplication. In this paper, we introduce a method for optimizing the performance of the deduplication system by improving the relevant key factors in CDC to find the optimal chunk cut-points and endorse a new hash function to generate a chunk's fingerprint, which significantly reduces the system's computational overhead.

3. Methodology

Data deduplication is needed to eliminate redundant data and reduce the required storage. It has been gaining increasing attention and has been evolving to meet the demand for storage cost saving, enhanced backup speed, and a reduced amount of data transmitted across a network. In this paper, we developed a CDC algorithm based on the frequency of bytes occurrence and a new hashing algorithm based on a mathematical triple hashing function.

3.1. Content-Defined Chunking

Content-defined chunking (CDC) is data deduplication chunking technology that sets the chunking breakpoint based on the content of the data, when a predefined breaking condition becomes true. It is used to eliminate the boundary shifting problem that fixed size chunking (static chunking) suffers from, as any modification to the stream of data by adding or removing one byte will lead to the generation of a different set of chunks, and these chunks will have different hash values, so it will be considered new data and will affect deduplication efficiency.

The most common systems of content-defined chunking by deduplication are as follows.

3.1.1. Basic Sliding Window (BSW) (Usually Known as Rabin CDC)

This is one of the legacy CDC chunking algorithms that use a fingerprint hashing function (Rabin fingerprint) as a breaking condition after setting three parameters that will be used for chunking (the sliding window size, the D-divisor, and the R-remainer, where D > R).

The main drawback of this algorithm is that, for each shifting in the window, a fingerprint will be generated and a condition will be checked (fingerprint mod D = R?) to set the chunk boundary. Such a calculation will consume an enormous amount of computational resources (rolling hash computation overhead), which affects the deduplication efficiency and throughput. Moreover, the D is a predefined average chunk size and not generated from the dataset itself, which leads to high chunk size variance [23]. Figure 1 illustrates the concept of the BSW algorithm.

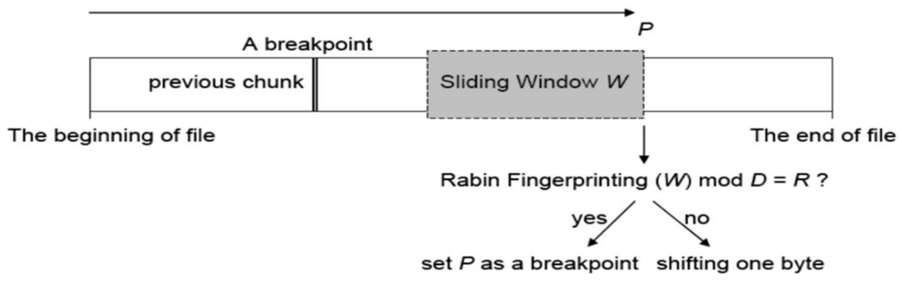

Figure 1. Concept of the basic sliding window (BSW) algorithm.

3.1.2. Two Threshold Two Divisors Chunking (TTTD)

This algorithm uses the same concept as BSW, but introduces four new parameters: the minimum size threshold (T_{min}), which is used to eliminate very small chunks, the maximum size threshold (T_{max}), used to eliminate very large chunks, the main divisor (D), and the secondary divisor (D'), which is half the value of the main divisor and used to find the breakpoint if the algorithm fails to find it with the main divisor.

The drawback of this algorithm is that the main divisor value is an estimated value and not related to the content of the data. In addition, the algorithm will not use the secondary divisor until reaching the T_{max} threshold, and in most cases the breakpoint found by the secondary divisor is very close to the T_{max} threshold. Such unnecessary calculations and comparisons are computationally expensive and affect the performance of the algorithm [23].

3.1.3. The Proposed Bytes Frequency Based Chunking (BFBC)

A new CDC algorithm depends on content analysis using statistical models generating a histogram of frequencies of occurrence of pair-bytes (pair-bytes distribution analysis) in the dataset to build a set of divisors to be used as chunking breaking points, as shown in Algorithm 1, which describes the chunking algorithm.

Algorithm 1 Chunking algorithm

```
Objective:  Divide the file into chunks based on $T_{min}$, List of Divisors, and $T_{max}$
Input:      File of any type or size
            $T_{min}$, $T_{max}$, List of Divisors
Output:     Number of variable sized chunks
Step1:      Set Breakpoint ← 0, Pbreakpoint ← 0
Step2:      Read File as array of bytes
            Set Length ← File size,
            Set Full Length ← File size
Step3:      If length equals zero
            Go to Step2
Step4:      If Length <= $T_{min}$
            Consider it as a chunk and send it to the Triple hash function, Algorithm (2)
            Length = Full Length—chunk length
Step5:      If Length between $T_{min}$ and $T_{max}$
            Search for the pair divisors bytes from breakpoint + $T_{min}$ until Full Length
                If found    Consider the chunk from Pbreakpoint until the divisor byte position
            Send the chunk to the Triple hash function, Algorithm (2)
            Breakpoint = chunk length +1
            Length = Full length − breakpoint
Step6:      If Length equals $T_{max}$
            Consider the chunk from Pbreakpoint until Full Length
            Send the chunk to the Triple hash function, Algorithm (2)
            Breakpoint= chunk length +1
            Length = Full Length—breakpoint
Step7:      If Length > $T_{max}$
            Search for the pair divisors bytes from breakpoint + $T_{min}$ until Pbreakpoint + $T_{max}$
                If found    Consider the chunk from Pbreakpoint until the breakpoint
            Send the chunk to the Triple hash function, Algorithm (2)
            Pbreakpoint = breakpoint
            Length = Full Length—breakpoint
Step8:      Go to step 3
```

Both BSW and TTTD suffer from the degradation of the chunking throughput due to Rabin's rolling hash function performance bottleneck. BFBC significantly improves chunking throughput and provides better deduplication efficiency by using a list of divisors generated from the statistical characteristics of the dataset itself, without the need to calculate the Rabin fingerprint for each cut-point judgment.

3.2. *Triple Hashing Algorithm*

Using the linear bounded sum of a string of non-repeatable zero bytes multiplied by a random sequence of numbers can produce hash values to be used as the chunk's fingerprint, as shown in Algorithm 2, because using three hashes to represent the chunk content can produce a combined fingerprint that has smaller hit rates of collision. The use of a linear sum of multiplication followed by a bounding operation has a low computational cost in comparison with the computational complexity of security hash functions (e.g., SHA1 and MD5).

The length of the proposed hash values to represent chunk fingerprints is very low (16 bits for each used hash function; a total of 48 bits for the three hashes), which is very small compared to the length of SHA1 (160 bits) or MD5 (128 bits); this will reduce the overhead information required to represent the hashing table. Below is Algorithm 2 for the proposed triple hashing function.

A full comparison between the proposed hashing function and the commonly used hashing functions (SHA1 and MD5) by other deduplication systems will be described and evaluated in the experimental section using hashing throughput and storage saving benchmarks.

Algorithm 2 Triple hash algorithm

```
Objective:  Generate three unique hash values for each chunk
Input:      Chunk as array of bytes
            Chunk Length
            HF1: array of integer contains 202 random values
            HF2: array of integer contains 202 random values
            HF2: array of integer contains 202 random values
Output:     Three hash values for the chunk
Step1:      Initialization
            Hash1 ← 3, Hash2 ← 37, Hash3 ← 17
            Li ← 0
Step2:      Compute three hash values for the chunk
            For I = 0 to Chunk Length-1 Do
                Li = Li +1
                If Li > 202   Li = Li - 202
                Hash1 = Hash1 + (HF1[Li] * Chunk[I])
                If Hash1 > 65535   Hash1 = Hash1 & 65535
                Hash2 = Hash2 + (HF2[Li] * Chunk[I])
                If Hash2 > 65535   Hash2 = Hash2 & 65535
                Hash3 = Hash3 + (HF3[Li] * Chunk[I])
                If Hash3 > 65535   Hash3 = Hash3 & 65535
            End For
```

3.3. Experimental Datasets

Two datasets with different characteristics were used to test system performance and efficiency. The first dataset consists of different versions of Linux source codes [24], from the Linux Kernel Archives, while the second dataset consists of 309 versions of SQLite [25]. Table 1 shows the characteristics of the used datasets.

Table 1. The characteristics of the tested datasets.

Dataset	Dataset 1	Dataset 2
Dataset Name	Linux Kernel	SQLite
Dataset Type	Linux source codes (3.16.57-4.18-rc6)	309 release of SQLite from version 1.0 to 3.33.0
No. of Files	450,441	212,741
Dataset Size (in MB)	6072	6596

4. The Proposed System

The proposed system focuses on a new CDC technique, used to produce chunks of variable size and determine chunk boundaries in the content by threshold breakpoints. Being aware of the problems in the CDC algorithm, in this paper, we propose a novel bytes frequency-based chunking (BFBC) technique. The BFBC technique reduces chunk size variation and maintains the DER and deduplication performance. It employs a sampling technique to determine the chunk boundaries based on the data content. The algorithm moves through the data stream byte by byte. If a data block satisfies certain pre-defined conditions (Divisors or T_{max}), it will be marked as a chunk cut-point. The data block between the current cut-point and its previous cut-point forms a resultant chunk. The BFBC technique consists of the following stages and is illustrated in Figure 2:

- Stage 1: Load the dataset.
- Stage 2: Divisors Analysis and Selection. The BFBC applies a content frequency scan to obtain the most frequent pair of bytes to be used as divisors (D) in the next stage.
- Stage 3: BFBC Chunking. The proposed BFBC algorithm is applied to obtain chunks. The input file is divided into variable-size chunks using multi divisors and two thresholds. The values of the divisors and the threshold are optimized.
- Stage 4: Triple Hashing, Indexing, and Matching. At this stage, a new hashing technique is suggested to enhance the matching process by generating the three hashes computed in the hashing stage. It consists of two steps:
    - Hash generation in which fingerprints of data content are created;
    - Hash judgement in which hashes are compared against already-stored hashes to find duplicated chunks (if matching occurs for the first hash value of the compared chunks, then it will compare the second and third).

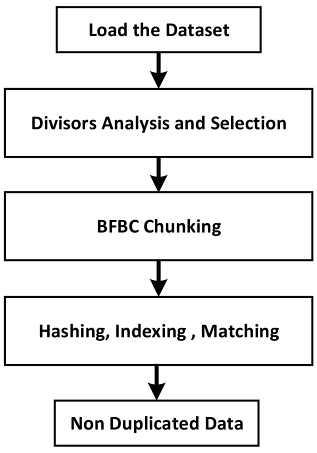

Figure 2. The proposed system.

4.1. Load the Dataset

In this stage, the system will start reading files from the dataset one by one, and then process these files as an array of bytes, preparing it to be scanned in the next stage.

4.2. Divisors Analysis and Selection

Byte distribution analysis is statistical analysis whereby a binary file is examined in terms of its byte constituents, since each dataset typically contains some bytes that are used constantly. This part of the proposed system was built to compute the frequency of the pair of bytes in the input file. The function will count the frequency of pairs of bytes in a file, i.e., how many times each pair of bytes is presented in the file to return maximum occurring pairs in the dataset. The function will traverse the given dataset byte by byte and store the frequencies or the number of times each pair occurs in a dataset. The function will then sort the output to find the list of pairs in descending order, so that the top pairs of bytes can be selected for the next stage of the chunking process. Table 2 shows the list of the top 10 pairs (divisors) generated by the divisors analysis and selection stage for Datasets 1 and 2, sorted in descending order. The number of divisors that yield the best DER will be discussed in the next section.

Table 2. Top 10 divisors ordered by frequency of occurrence for Datasets 1 and 2.

Dataset 1 (Linux Versions) Top 10 Pairs of Bytes Occurrence				**Dataset 2 (SQLite Versions) Top 10 Pairs of Bytes Occurrence**			
Pair Bytes Value Symbols		**Pair Bytes Value Description**	**No. of Occurrence**	**Pair Bytes Value Symbols**		**Pair Bytes Value Description**	**No. of Occurrence**
Space	Space	Space—Space	213,150,255	NUL	NUL	Null—Null	643,401,100
LF	HT	Line Feed—Horizontal Tab	98,879,678	Space	Space	Space—Space	335,181,944
HT	HT	Horizontal Tab—Horizontal Tab	83,978,466	LF	Space	Line Feed—Space	64,205,375
i	n	Lowercase i—Lowercase n	55,197,438	Space	t	Space—Lowercase t	57,427,195
e	Space	Lowercase e—Space	54,008,419	e	Space	Lowercase e—Space	50,525,015
;	LF	Semicolon—Line Feed	52,619,629	0	0	Zero—Zero	47,909,305
,	Space	Comma—Space	47,951,864	s	e	Lowercase s—Lowercase e	41,087,776
d	e	Lowercase d—Lowercase e	44,203,960	t	Space	Lowercase t—Space	38,889,278
r	e	Lowercase r—Lowercase e	42,285,606	i	n	Lowercase i—Lowercase n	33,847,741
t	Space	Lowercase t—Space	40,881,129	,	Space	Comma—Space	32,878,767

4.3. BFBC Chunking

The main task of this phase is to partition the input file (stream of bytes) into small and non-overlapped chunks using the new BFBC technique. It will determine the chunk boundary or the breakpoint by a certain group of divisors, which depend on the contents of the dataset. It is based on the characteristic of the original file as a data container and defines the boundaries that are based on the occurrence of pairs of bytes in the dataset. The condition is used in BFBC either to find one of the listed multi divisrs D (generated in the previous stage) after T_{min} or to reach the T_{max} threshold value if determination of the breakpoint failed using the divisors conditions. BFBC guarantees a minimum and maximum chunk size. Minimum and maximum size limits are used to split a file into chunks when D is not found. There are three main parameters that need to be pre-configured: the minimum threshold (Tmin), the list of divisors (D), and the maximum threshold (T_{max}), as described in Table 3.

Table 3. The purpose of chunking parameters.

Parameter	Purpose
T_{min} (minimum threshold)	To reduce the number of very small chunks
Divisors (D)	To determine breakpoints
T_{max} (maximum threshold)	To reduce the number of very large chunks

When the chunking method depends on the format of the file, the deduplication method can provide the best redundancy detection ratio compared with the fixed-size and other variable-size chunking methods.

Chunking steps are as follows:

a. Scan the sequence of bytes starting from the last chunk boundary and apply a minimum threshold to the chunk sizes (T_{min}).
b. At each position after T_{min}, check the current pair of bytes with the list of divisors to look for matches.
c. If a D-match is found before reaching the threshold T_{max}, use that position as a breakpoint.
d. If the search for a D-match fails and the T_{max} threshold is reached (without finding a D-match), use the current position (threshold T_{max}) as a breakpoint.

Tail chunks result for the following reasons:

- The size of the file is smaller than T_{min};
- The fragment, from the last breakpoint to the end of the file, is smaller than T_{min};
- The algorithm cannot find any breakpoint from the last breakpoint to the end of the file, even if the size of the chunk is larger than T_{min}.

Figure 3 shows the chunk types generated by the chunking stage. Figure 4 illustrates the chunking technique used in the proposed system.

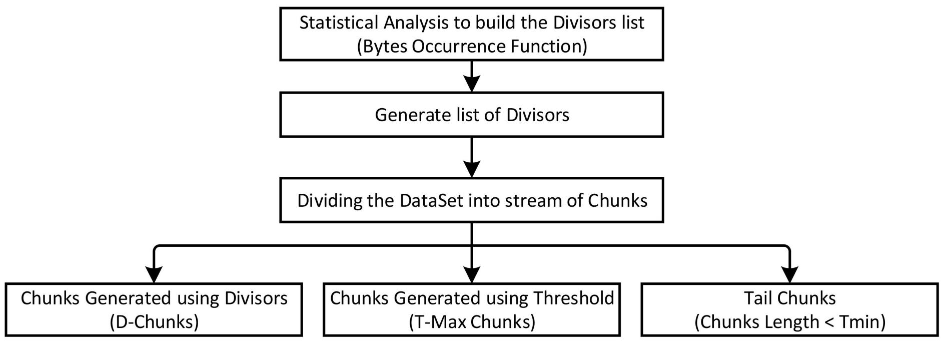
Statistical Analysis to build the Divisors list
(Bytes Occurrence Function)
Generate list of Divisors
Dividing the DataSet into stream of Chunks
Chunks Generated using Divisors
(D-Chunks)
Chunks Generated using Threshold
(T-Max Chunks)
Tail Chunks
(Chunks Length < Tmin)

Figure 3. Types of chunks generated from the chunking stage.

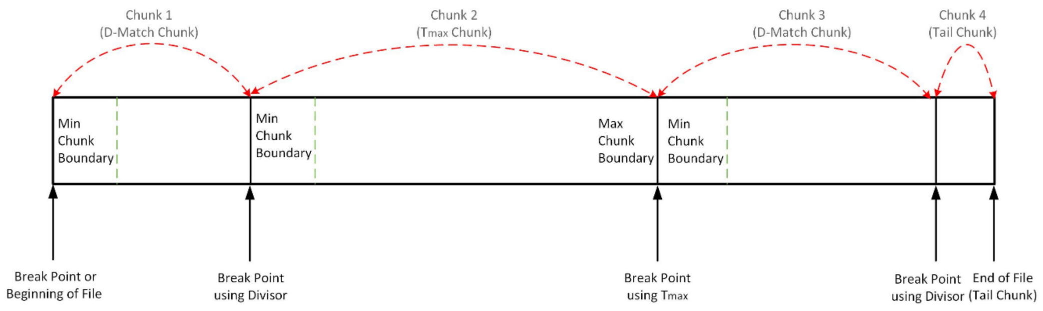

Figure 4. Chunking technique of the proposed system.

4.4. Triple Hashing, Indexing, and Matching Stage

Legacy deduplication systems suffer from high computational power and disk space requirements and waste time solving the collision problem. In this work, a new hashing technique is proposed to save resources and reduce processing time.

The hashing part of the system uses a new simple hashing function to compute three hash values for each chunk. Therefore, each chunk produced from the chunking stage is sent to the proposed hash function to generate three hash values to describe the contents of chunks. Each hash is generated by a mathematical function with a size of 16 bits. The total number of bits needed to store these three hashes is 48. Traditional hashing functions (SHA1 and MD5) used by other content-defined chunking methods consume a substantial amount of processing resources to calculate hash values, and they utilize much more storage space to store these hash values. The number of bits needed to store our proposed hashes is less than the number of bits needed to save the hash value using SHA1 (160 bit) and MD5 (128 bit).

For each chunk, a chunk ID is created, which includes chunk size, divisor type, and the three generated hashes. To find the duplicated chunks, if the chunk size, divisor type, and the first hash of the two compared chunks match, then the second hash of the two chunks are compared, followed by the third. This cascade comparison will reduce the time needed to compare the chunks by eliminating the byte-to-byte comparison that is needed to compare chunks. If they are found to be identical, the chunk's pointer in the metadata table is updated to the already-existing chunk by incrementing the chunk reference count for that chunk, and the new (duplicate) chunk is released. Otherwise, the new (non-duplicated) chunk is saved in the unique data container, its chunk reference is saved in the metadata table, and the three hashes are saved in a temporary hash index table for further processing. Figure 5 shows the steps of this stage.

After deduplication, three types of data are stored:

- Unique data. Non-duplicated chunks are produced by the algorithm.
- Metadata. To rebuild the dataset again, address information related to the chunks needs to be stored. Regardless of whether chunks contain unique data or not, the related metadata needs to be available for future retrieving stage. Accordingly, the total metadata number equals the total number of chunks.
- Hash index. Hash values for each non-duplicated chunk are stored as the chunk's fingerprint for use in detecting duplicated chunks. Each record within the hash index table is considered as a chunk ID. Table 4 shows an example of a hash index table.

Table 4. Example of hashing index table.

Chunk Size (In Byte)	Pair Divisor (In Decimal)		Hash1	Hash2	Hash3
166	105	110	62,724	944	3581
148	32	32	65,132	23,057	51,437
138	105	110	45,542	52,089	33,677
432	32	32	53,535	60,344	15,897
369	105	110	8070	25,948	50,716
147	32	32	52,628	46,251	3768

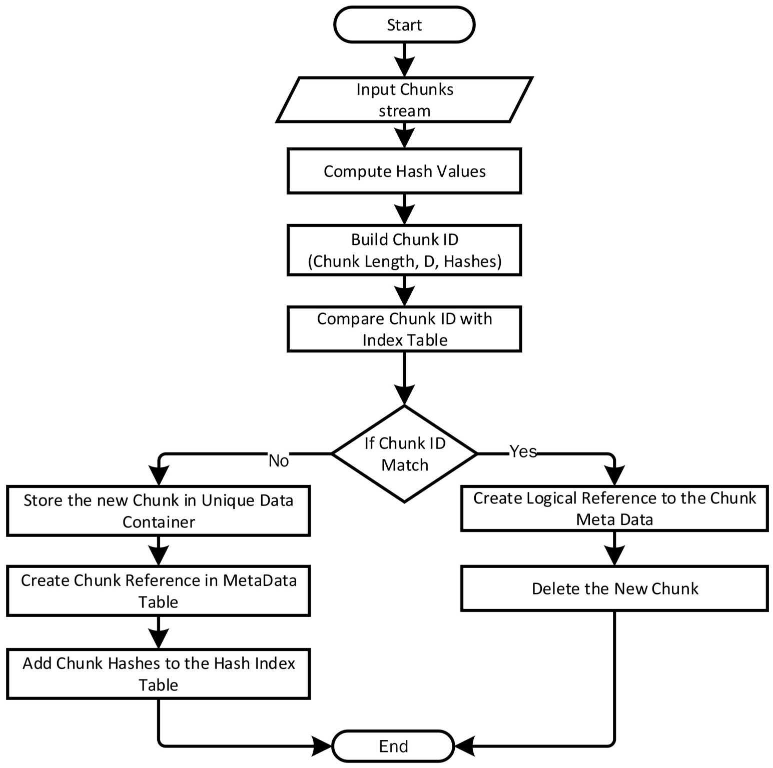
Start
Input Chunks stream
Compute Hash Values
Build Chunk ID (Chunk Length, D, Hashes)
Compare Chunk ID with Index Table
If Chunk ID Match
No
Yes
Store the new Chunk in Unique Data Container
Create Chunk Reference in MetaData Table
Add Chunk Hashes to the Hash Index Table
Create Logical Reference to the Chunk Meta Data
Delete the New Chunk
End

Figure 5. Hashing, indexing, and matching stage.

5. Experiment

5.1. Experimental Setup

We built the deduplication system from scratch including chunking, hashing, and matching stages using C# language. The configuration of the computer used for the experiment is described as follows:

- CPU: Intel Core i7-3820QM @ 2.70 GHz 4-core processor;
- RAM: 16 GB DDR3;
- Disk: 1TB PCIe SSD;
- Operating system: Windows 10 64-bit.

5.2. Experimental Results

In this section, we will analyze some properties of the algorithm, mainly, chunk size, divisors selection, and the proposed hashing function effects. This section describes our evaluation of storage space saving and describes the relationship between the storage saving and the two chunking parameters (no. of divisors and chunk size). We hope to propose a more precise approach to improve the deduplication ratio. It is shown that the ratio of the storage capacity can be reduced if data deduplication techniques are applied.

5.2.1. Choosing the Optimum Chunk Size (Optimizing Storage Saving by Setting the Expected Chunk Size)

Chunk size has a direct impact on the deduplication ratio. Chunk size needs to be optimized to achieve the best results. Small T_{min}/T_{max} values detect more duplicated chunks, but they affect the metadata size. Large T_{min}/T_{max} values decrease the deduplication ratio because they produce large chunks, which in turn further reduces redundant data detection. Table 5 shows the dataset size after deduplication for different values of T_{min}/T_{max}.

Table 5. Dataset size after deduplication using differnet T_{min}/T_{max} values.

	Dataset 1 (Linux Versions)	**Dataset 2 (SQLite Versions)**
$\mathbf{T_{min}–T_{max}}$	**Dataset Size after Deduplication (MB)**	**Dataset Size after Deduplication (MB)**
128–256	1239	219.2
128–512	1224	214.9
128–1024	1225	220.9
128–2048	1225	221.5
128–4096	1225	221.9
128–8192	1226	222.2
256–512	1525	280.9
256–1024	1520	285.1
256–2048	1521	285.5
256–4096	1521	285.8
256–8192	1521	286.2
512–1024	1875	390.3
512–2048	1873	390.0
512–4096	1874	390.2
512–8192	1874	390.4
1024–2048	2163	518.4
1024–4096	2163	517.6
1024–8192	2163	517.8
2048–4096	2382	626.1
2048–8192	2382	625.9
4096–8192	2550	701.1

The experimental results are presented in Figure 6. The *x*-axis represents the chunk size, and the *y*-axis represents the dataset size after deduplication. As chunk size increases (T_{min}/T_{max}), the deduplication ratio gradually degrades.

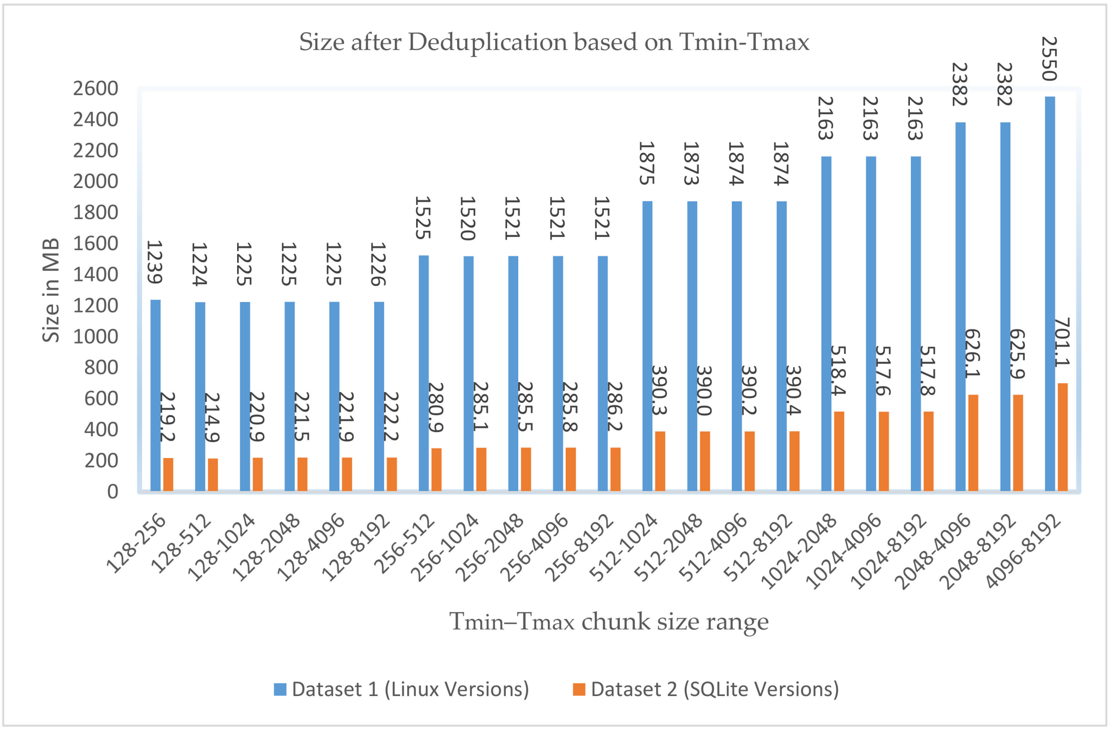

Figure 6. The relationship between chunk size ($T_{min}–T_{max}$) and dataset size after deduplication.

The results show that storage size after duplication elimination is optimal by setting the chunk size between 128 and 512 bytes for both datasets. Deduplication at that chunk size reduced 5.93 GB of data to 1.19 GB in Dataset 1 and 6.44 GB to 214.9 MB in Dataset 2. Given the significant advantage shown of small block sizes, the results illustrate why T_{min}/T_{max} plays an important role in deduplication.

5.2.2. Chunk Distribution Based on the Number of Divisors

The efficiency of the proposed system is based on the ratio of chunks, which was generated using the list of proposed divisors. Figure 7 and Table 6 show the chunk distribution based on the number of divisors in Dataset 1.

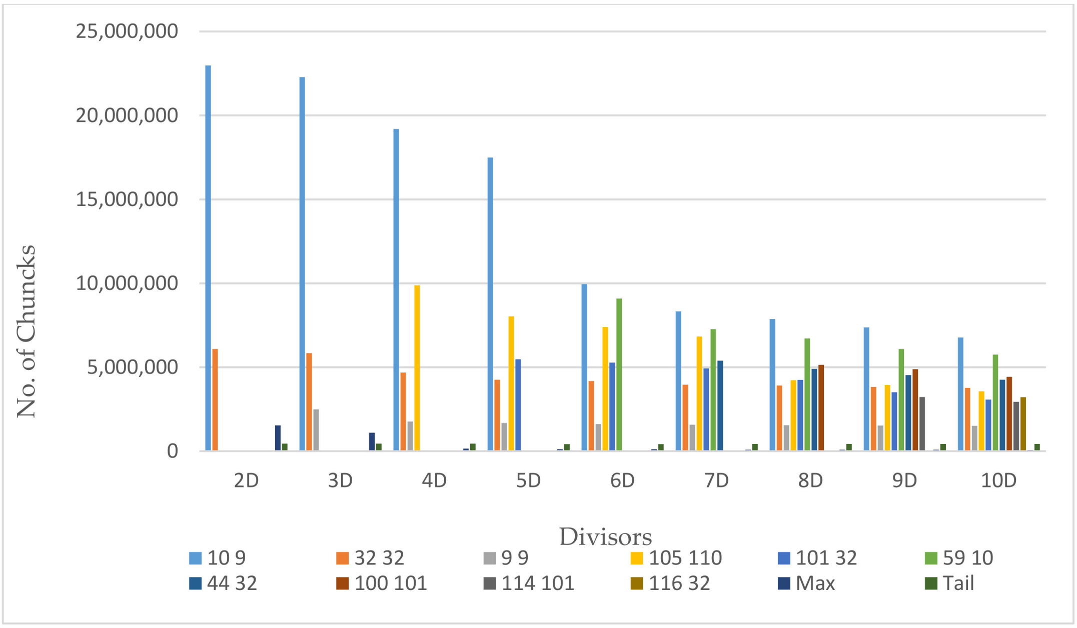

Figure 7. Chunk distribution based on the number of divisors (Dataset 1).

Table 6. Chunk occurrence matrix based on the number of divisors (Dataset 1).

Divisor Type	2D	3D	4D	5D	6D	7D	8D	9D	10D
10 9	22,973,671	22,284,893	19,199,186	17,492,783	9,953,964	8,330,810	7,873,710	7,374,764	6,779,877
32 32	6,092,243	5,837,677	4,685,335	4,255,047	4,174,575	3,958,998	3,913,365	3,829,970	3,769,410
9 9		2,486,258	1,770,387	1,684,706	1,617,595	1,571,559	1,552,364	1,528,012	1,512,120
105 110			9,884,948	8,039,422	7,400,438	6,836,950	4,235,299	3,943,700	3,566,692
101 32				5,474,256	5,276,818	4,928,659	4,251,855	3,513,049	3,072,207
59 10					9,094,685	7,276,476	6,718,100	6,085,059	5,758,142
44 32						5,384,122	4,905,076	4,539,605	4,258,704
100 101							5,144,276	4,883,201	4,431,435
114 101								3,228,453	2,943,306
116 32									3,216,933
Max	1,543,861	1,097,639	161,528	117,745	117,745	89,660	87,154	85,693	72,411
Tail	450,316	450,316	450,316	430,287	430,287	431,530	431,955	433,090	433,136

Table 7 shows that most chunks were generated using divisors, while the ratio of the chunks generated using T_{max} are minimal. According to the results of our experiments, we found that more than 98% of the total chunks were determined by the divisor when the chunk size was 128–512 bytes, which clearly shows the efficiency of the propsoed chunking algorithm.

Table 7. Divisors, T_{max}, and tail chunk ratio for Dataset 1 based on selected T_{min}–T_{max} values.

Chunking Category (T_{min}–T_{max})	Total No. of Chunks	No. of Chunks Generated Using Divisors	No. of Chunks Generated Using T_{max}	No. of Tail Chunks	Divisors Chunks Ratio	T_{max} Chunks Ratio	Tail Chunks Ratio
128–256	39,908,873	39,260,746	197,811	450,316	98.38%	0.50%	1.13%
128–512	39,814,373	39,291,646	72,411	450,316	98.69%	0.18%	1.13%
128–1024	39,776,247	39,294,163	31,768	450,316	98.79%	0.08%	1.13%
128–2048	39,759,163	39,294,699	14,148	450,316	98.83%	0.04%	1.13%
128–4096	39,751,564	39,294,788	6460	450,316	98.85%	0.02%	1.13%
128–8192	39,747,969	39,294,796	2857	450,316	98.86%	0.01%	1.13%
256–512	20,921,695	20,395,370	76,009	450,316	97.48%	0.36%	2.15%
256–1024	20,883,699	20,401,331	32,052	450,316	97.69%	0.15%	2.16%
256–2048	20,866,807	20,402,286	14,205	450,316	97.77%	0.07%	2.16%
256–4096	20,859,297	20,402,512	6469	450,316	97.81%	0.03%	2.16%
256–8192	20,855,734	20,402,561	2857	450,316	97.83%	0.01%	2.16%
512–1024	10,814,367	10,330,633	33,418	450,316	95.53%	0.31%	4.16%
512–2048	10,797,458	10,332,812	14,330	450,316	95.70%	0.13%	4.17%
512–4096	10,790,036	10,333,241	6479	450,316	95.77%	0.06%	4.17%
512–8192	10,786,472	10,333,295	2861	450,316	95.80%	0.03%	4.17%
1024–2048	5,592,898	5,127,700	14,882	450,316	91.68%	0.27%	8.05%
1024–4096	5,585,435	5,128,612	6507	450,316	91.82%	0.12%	8.06%
1024–8192	5,581,935	5,128,756	2863	450,316	91.88%	0.05%	8.07%
2048–4096	2,938,586	2,481,627	6643	450,316	84.45%	0.23%	15.32%
2048–8192	2,935,094	2,481,902	2876	450,316	84.56%	0.10%	15.34%
4096–8192	1,611,792	1,158,565	2911	450,316	71.88%	0.18%	27.94%

5.2.3. The Impact of the Number of Divisors Selected on the Data Duplication Ratio

In this subsection, we present results from the evaluation of our deduplication technique based on experiments and analysis. The space savings achievable with deduplication are shown to indicate the usefulness of deduplication with respect to our target workload.

Figure 8 shows the deduplication behavior when selecting a different number of divisors. The x-axis represents the number of divisors selected, while the y-axis represents the storage size after deduplication.

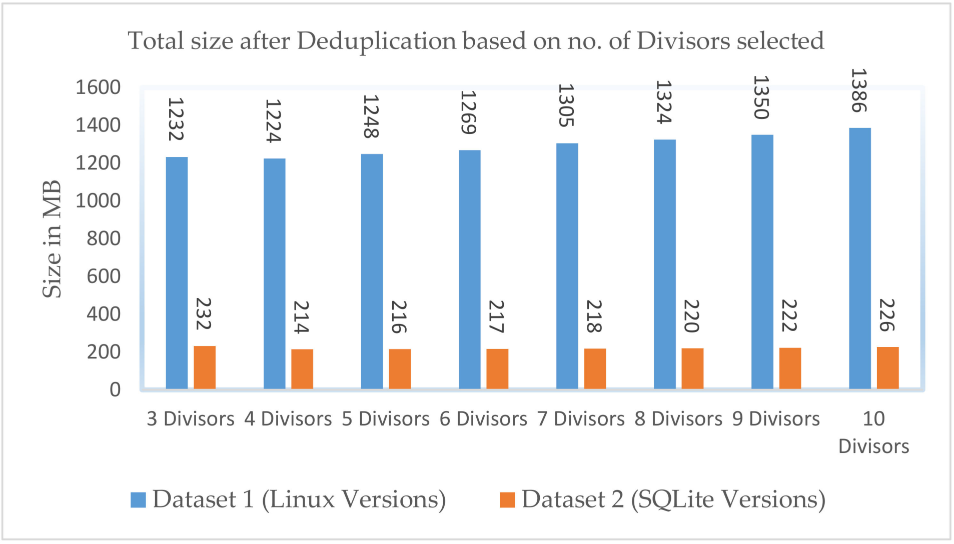

Figure 8. Storage size after deduplication based on the number of divisors.

To determine the optimum number of divisors for the experiments, the proposed system was tested using a different range of divisors. To obtain a higher deduplication ratio, we selected an appropriate number of divisors.

After examining the pair bytes occurrence (divisors), we found that a 128–512 chunk size will yield the best deduplication ratio, where the number of pairs is between 3 and 10. According to Figure 8, when the number of divisors is 4, we yielded the smallest total size after deduplication (highest storage gain) for both datasets. Figure 9 shows that the storage gain varies among different numbers of divisors. Accordingly, the storage gain after deduplication (deduplication space savings) reached 76.98% for Dataset 1 and 96.76% for Dataset 2 when the number of divisors was four.

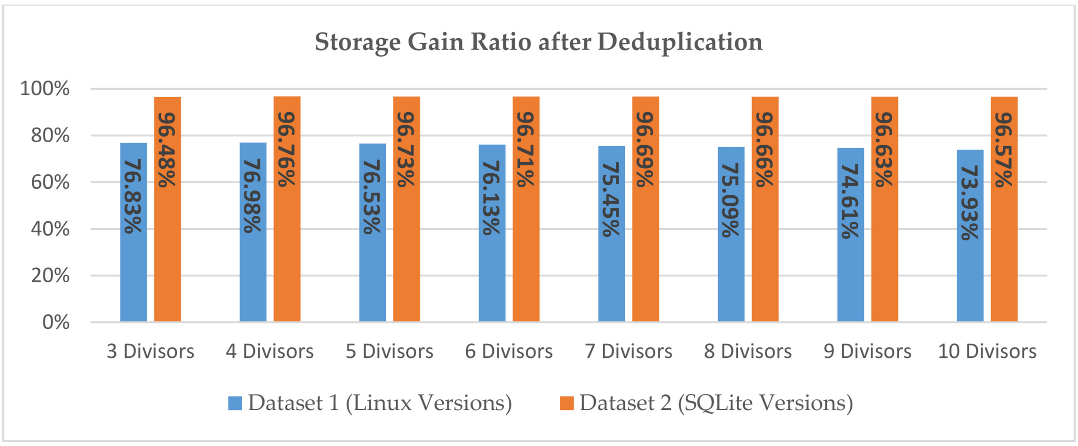

Figure 9. Storage gain ratio after deduplication based on the number of divisors.

Table 8 shows a sample of the number of duplicate chunks in Dataset 1 for each chunk size, with specific hash values in terms of the number of references to each block in the file system after deduplication. At the very peak, some chunks were duplicated more than 4000 times in Dataset 1. Each of these chunks individually represents an enormous amount of space that was wasted storing duplicate data. Overall, these data serves to show the possibility of space savings from deduplication.

Table 8. Examples of numbers of duplicate chunks corresponding to chunk sizes with specific hash values (Dataset 1).

Chunk Size (In Byte)	Divisor (In Decimal)		Hash1	Hash2	Hash3	No. of Duplicated Chunks
365	32	32	11,413	20,379	56,737	4753
280	32	32	21,996	52,614	5429	4227
409	32	32	47,374	4130	36,809	3690
257	32	32	23,630	9669	17,229	3593
286	105	110	50,852	51,897	47,689	3495
313	105	110	54,191	58,279	1797	2883
380	105	110	34,997	55,857	5919	2856
292	10	9	1674	49,658	23,838	2824
292	10	9	2938	51,192	26,116	2824
380	105	110	3978	55,689	16,358	2636

5.2.4. Proposed Chunking Algorithim Efficiency

Chunking is a time- and resource-consuming process because it has to navigate the entire file byte by byte to find the cut-points. The processing time and resource utilization of the chunking stage is entirely dependent on the conditions of the chunking algorithms breaking the file. In this experiment, we compared the proposed system with BSW and TTTD algorithms, as both have static divisor D values. In BFBC, the system will generate the best list of divisors dynamically based on the statistical analysis of the dataset content to discover maximum redundancy and will find the duplicated data faster than other approaches. According to the results, BFBC is about 10 times faster than BSW and three times faster than TTTD, which leads to a significant increase in chunking throughput, as shown in Table 9 and Figures 10 and 11.

$$\textit{Chunking Algorithm Throughput} = \textit{chunking Computational overhead} = \frac{\textit{Processed Data in MB}}{\textit{Time in Second}} \quad (1)$$

Table 9. Chunking time and thoughput for the proposed bytes frequency-based chunking (BFBC) chunking compared with BSW and two thresholds two divisors (TTTD).

	Chunking Time (Second)			**Chunking Throughput (MB/s)**		
	BSW	**TTTD**	**BFBC**	**BSW**	**TTTD**	**BFBC**
Dataset 1 (Linux Versions)	557	189	58	10.9	32.1	104.7
Dataset 2 (SQLite Versions)	614	210	60	10.7	31.4	109.9

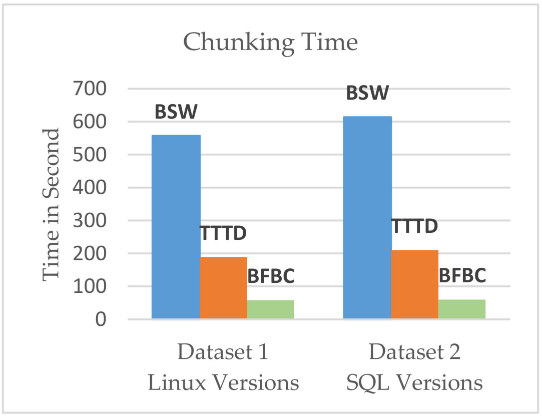

Figure 10. Chunking time comparison.

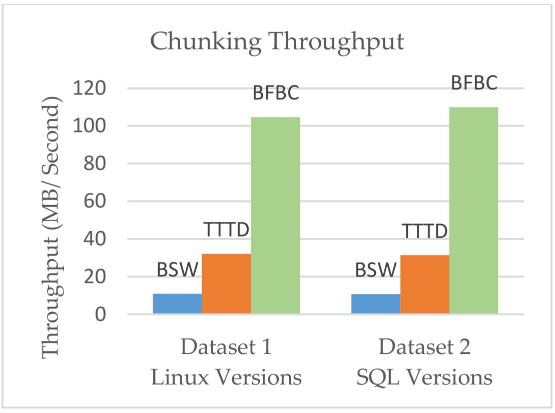

Figure 11. Chunking throughput comparison.

5.2.5. The Impact of the Proposed Hashing Function

A. Impact on Storage Utilization: The proposed mathematical triple hash function to generate the chunk's fingerprint has a direct impact on storing the hashes in the index table. Each fingerprint requires 48 bits (6 bytes), while traditional hash functions MD5 and SHA1 require 128 bits (16 bytes) and 160 bits (20 bytes), respectively. Table 10 and Figure 12 show the impact of the hash algorithms on the storage size required for storing fingerprints in the index table computed by Equation (2).

Considering that, the total number of chunks after deduplication was 8,255,963 for Dataset 1 and 1,066,606 for Dataset 2.

$$Hashing\ index\ table\ size = No.\ of\ Bytes\ per\ Fingerprint \times Total\ No.\ of\ Chunks\ after\ DD \quad (2)$$

Table 10. Required storage size for storing fingerprints (hashing index table).

	Hashing Index Table Size (MB)		
	SH1	**MD5**	**Triple Hash Function**
Dataset 1 (Linux Versions)	157.4	126.0	47.2
Dataset 2 (SQLite Versions)	20.3	16.2	6.1

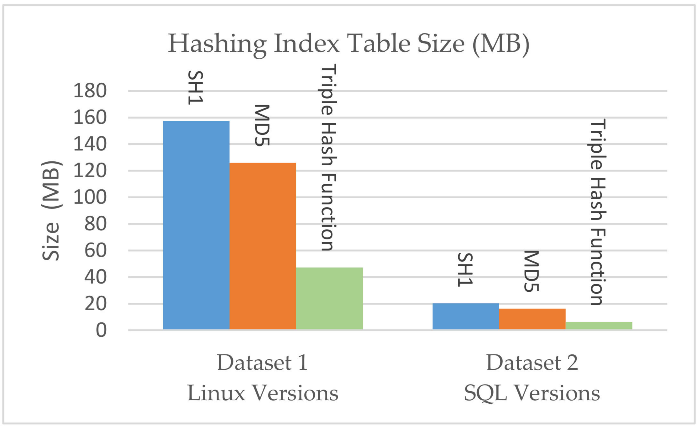

Figure 12. Hashing Index Table size comparison.

B. Impact on Computational Overhead: The proposed triple hash algorithm uses a simple mathematical equation compared with the traditional hashing functions (SHA1 and MD5) used by other content-defined chunking, which consumes substantial processing resources and leads to heavy CPU overhead when calculating hash values. Table 11 and Figures 13 and 14 show the impact of hash algorithms on the hashing stage time and throughput, computed by Equation (3).

Table 11. Hashing time and thoughput for the proposed triple hash function compared with SHA1 and MD5.

	Hashing Time (Seconds)			Hashing Throughput (MB/s)		
	SH1	**MD5**	**Triple Hash Function**	**SH1**	**MD5**	**Triple Hash Function**
Dataset 1 (Linux Versions)	462.9	424.2	87.5	13.1	14.3	69.4
Dataset 2 (SQLite Versions)	533.0	493.0	114.0	12.4	13.4	57.9

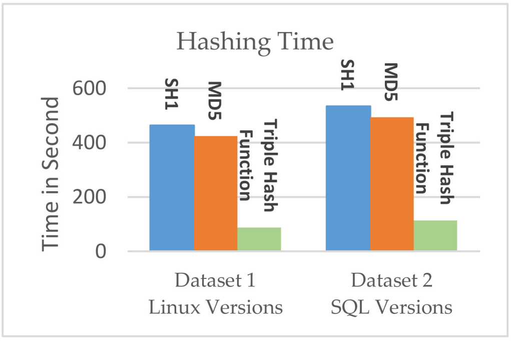

Figure 13. Hashing Time comparison.

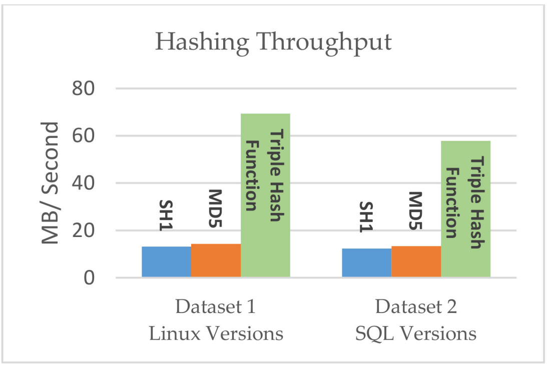

Figure 14. Hashing Throughput comparison.

According to the results below, for both datasets, the proposed algorithm requires the least hashing time, which leads to a better throughput compared to the traditional hashing algorithms.

$$Hashing\ Algorithm\ Throughput = Hashing\ Computational\ overhead = \frac{Processed\ Data\ in\ MB}{Time\ in\ Second} \quad (3)$$

5.2.6. Data Size after Deduplication and the Deduplication Elimination Ratio (DER)

The performances of the BSW, TTTD, and our proposed solution were compared in terms of size after deduplication and the DER (computed by Equation (4)). Results presented in Table 12 and Figures 15 and 16 clearly show that the proposed chunking alogrithm provides increased storage saving and an improved DER compared with other deduplication methods.

$$Deduplication\ Elimination\ Ratio\ (DER) = \frac{Input\ Datas\ size\ before\ Dedulplication\ in\ MB}{Output\ Data\ size\ after\ Deduplication\ in\ MB} \quad (4)$$

Table 12. Total size after deduplication and the deduplication elimination ratio (DER).

	Size after Deduplication in MB			**Deduplication Elimination Ratio (DER)**		
	BSW	**TTTD**	**BFBC**	**BSW**	**TTTD**	**BFBC**
Dataset 1 (Linux Versions)	1928	1539	1224	3.15	3.95	4.96
Dataset 2 (SQLite Versions)	489	396	214	13.49	16.66	30.82

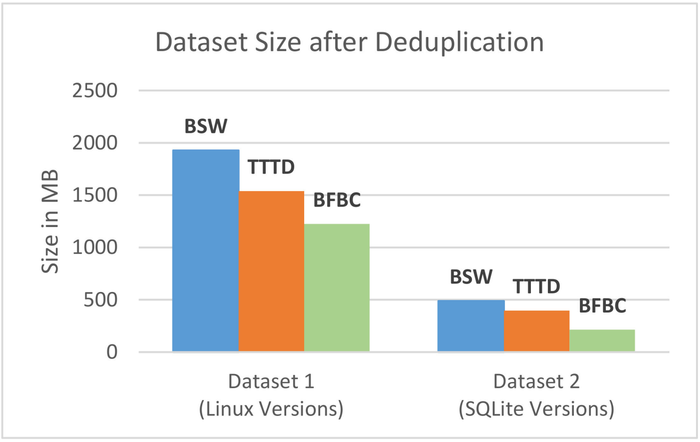

Figure 15. Dataset size after deduplication comparison.

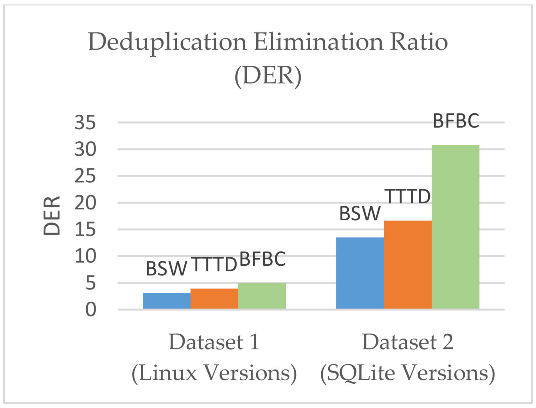

Figure 16. DER comparison.

The objective of our experiments is to compare the performance of the proposed BCFB chunking algorithm with that of the BSW and TTTD chunking algorithms, and compare the proposed triple hashing algorithm with SHA1 and MD5. BFBC was shown to effectively improve the deduplication throughput performance and, with the help of the triple hashing function, reduce computation time dramatically, as shown in the experiment results.

6. Conclusions and Future Work

In this paper, a combination of two new approaches for chunking and hashing algorithms provides impressive storage efficiency by reducing space usage and optimizing CPU throughput. The first approach is the Bytes Frequency-Based Chunking (BFBC), which made use of bytes frequency occurrence information from the dataset to improve data deduplication gain. This is based on the statistical bytes frequency, which indicates highly frequent pairs of bytes within the dataset. We demonstrated that our proposed approach can make full use of the list of divisors, which is the core component that, by designing a break condition for cut-point identification using a list of predefined divisors, enhances the performance of the chunking algorithm. The second approach is the proposed triple hash algorithm using a mathematical function that generates short fingerprints, which has a direct impact on the index table size and hashing throughput.

The experimental results show that chunking, using a list of divisors generated based on the content of the dataset, and setting the expected chunk size thresholds (T_{min}–T_{max}) to 128–512 bytes in the BFBC chunking algorithm can effectively improve the deduplication throughput performance. BFBC is 10 times faster than BSW and approximately three times faster than TTTD, and the proposed triple hash function is five times faster than SHA1 and MD5.

However, there are possible limitations. First, system efficiency may be affected if the dataset content has a low ratio of similarity (e.g., contains a high number of compressed images or audio files); in this case, systems will face performance degradation due to the enormous variance in the content of the dataset. Another limitation is that running the system using a large dataset, to reduce the possibility of hash collision, will require an increase in the size of the hashes that represent the fingerprint; this will increase the hash index table size and computational overhead.

In the future, we will study the option of building an automated method that generates the list of divisors based on the percentage of cumulative pair bytes occurrence (e.g., that automatically generates a list of divisors based on a 20% cumulative occurrence of pair bytes) and analyses system behavior when triplets of bytes (or larger byte groups) are used instead of pairs of bytes in the divisors analysis and selection stage.

Author Contributions: The model was proposed by A.S.M.S., who collected experimental data and performed the analysis; L.E.G. provided guidance for this paper and was the research advisor. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare that there is no conflict of interest.

References

1. Xia, W.; Jiang, H.; Feng, D.; Douglis, F.; Shilane, P.; Hua, Y.; Fu, M.; Zhang, Y.; Zhou, Y. A comprehensive study of the past, present, and future of data deduplication. *Proc. IEEE* **2016**, *104*, 1681–1710. [CrossRef]
2. Kambo, H.; Sinha, B. Secure Data Deduplication Mechanism Based on Rabin CDC and MD5 in Cloud Computing Environment. In Proceedings of the 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, India, 19–20 May 2017; IEEE: New York, NY, USA, 2017.
3. Manogar, E.; Abirami, S. A Study on Data Deduplication Techniques for Optimized Storage. In Proceedings of the 2014 Sixth International Conference on Advanced Computing (ICoAC), Chennai, India, 17–19 December 2014; IEEE: New York, NY, USA, 2014.
4. Lu, G.; Jin, Y.; Du, D.H. Frequency based chunking for data de-duplication. In Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Miami Beach, FL, USA, 17–19 August 2010; IEEE: New York, NY, USA, 2010.
5. Puzio, P.; Molva, R.; Önen, M.; Loureiro, S. Block-level de-duplication with encrypted data. *Open J. Cloud Comput.* **2014**, *1*, 10–18.
6. Zhang, Y.; Jiang, H.; Feng, D.; Xia, W.; Fu, M.; Huang, F.; Zhou, Y. AE: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication. In Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM), Hong Kong, China, 26 April–1 May 2015; IEEE: New York, NY, USA, 2015.
7. Xia, W.; Jiang, H.; Feng, D.; Tian, L.; Fu, M.; Wang, Z. P-dedupe: Exploiting parallelism in data deduplication system. In Proceedings of the 2012 IEEE Seventh International Conference on Networking, Architecture, and Storage, Xiamen, China, 28–30 June 2012; IEEE: New York, NY, USA, 2012.
8. Zhang, Y.; Wu, Y.; Yang, G. Droplet: A distributed solution of data deduplication. In Proceedings of the 2012 ACM/IEEE 13th International Conference on Grid Computing, Beijing, China, 20–23 September 2012; IEEE: New York, NY, USA, 2012.
9. Zhang, C.; Qi, D.; Cai, Z.; Huang, W.; Wang, X.; Li, W.; Guo, J. MII: A novel content defined chunking algorithm for finding incremental data in data synchronization. *IEEE Access* **2019**, *7*, 86932–86945. [CrossRef]
10. Venish, A.; Sankar, K.S. Study of chunking algorithm in data deduplication. In Proceedings of the International Conference on Soft Computing Systems; Springer, New Delhi, India; 2016.
11. Kumar, N.; Antwal, S.; Samarthyam, G.; Jain, S.C. Genetic optimized data deduplication for distributed big data storage systems. In Proceedings of the 2017 4th International Conference on Signal Processing, Computing and Control (ISPCC), Solan, India, 21–23 September 2017; IEEE: New York, NY, USA, 2017.
12. Ha, J.-Y.; Lee, Y.-S.; Kim, J.-S. Deduplication with block-level content-aware chunking for solid state drives (SSDs). In Proceedings of the 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, Zhangjiajie, China, 13–15 November 2013; IEEE: New York, NY, USA, 2013.
13. Yu, C.; Zhang, C.; Mao, Y.; Li, F. Leap-based content defined chunking—theory and implementation. In Proceedings of the 2015 31st Symposium on Mass Storage Systems and Technologies (MSST), Santa Clara, CA, USA, 30 May–5 June 2015; IEEE: New York, NY, USA, 2015.
14. Attarde, D.; Vijayan, M.K. Extensible Data Deduplication System and Method. U.S. Patent 8,732,133, 20 May 2014.
15. Zhou, B.; Zhang, S.; Zhang, Y.; Tan, J. A bit string content aware chunking strategy for reduced CPU energy on cloud storage. *J. Electr. Comput. Eng.* **2015**, *2015*, 242086. [CrossRef]
16. Wang, L.; Dong, X.; Zhang, X.; Guo, F.; Wang, Y.; Gong, W. A logistic based mathematical model to optimize duplicate elimination ratio in content defined chunking based big data storage system. *Symmetry* **2016**, *8*, 69. [CrossRef]
17. Kaur, R.; Chana, I.; Bhattacharya, J. Data deduplication techniques for efficient cloud storage management: A systematic review. *J. Supercomput.* **2018**, *74*, 2035–2085. [CrossRef]

18. Zhang, Y.; Feng, D.; Jiang, H.; Xia, W.; Fu, M.; Huang, F.; Zhou, Y. A fast asymmetric extremum content defined chunking algorithm for data deduplication in backup storage systems. *IEEE Trans. Comput.* **2017**, *66*, 199–211. [CrossRef]
19. Nie, J.; Wu, L.; Liang, J. Optimization of De-duplication Technology Based on CDC Blocking Algorithm. In Proceedings of the 2019 12th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), Suzhou, China, 19–21 October 2019; IEEE: New York, NY, USA, 2019.
20. Ni, F. *Designing Highly-Efficient Deduplication Systems with Optimized Computation and I/O Operations*; University of Texas at Arlington: Arlington, TX, USA, 2019.
21. Xia, W.; Zou, X.; Jiang, H.; Zhou, Y.; Liu, C.; Feng, D.; Hua, Y.; Hu, Y.; Zhang, Y. The Design of Fast Content-Defined Chunking for Data Deduplication Based Storage Systems. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *31*, 2017–2031. [CrossRef]
22. Maqdah, R.G.; Tazda, R.G.; Khakbash, F.; Marfsat, M.B.; Asghar, S.A. CA-Dedupe: Content-aware deduplication in SSDs. *J. Supercomput.* **2020**, *76*, 8901–8921.
23. Chang, B. A running Time Improvement for Two Thresholds Two Divisors Algorithm. Master's Thesis, San Jose State University, San Jose, CA, USA, 2009.
24. Linux, The Linux Kernel Archives. Available online: http://kernel.org/ (accessed on 4 March 2020).
25. D.RichardHipp, SQLite Kernel Archives. Available online: https://www.sqlite.org/chronology.html (accessed on 1 October 2020).