Journal of King Saud University – Computer and Information Sciences 34 (2022) 4669–4678

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

Lightweight hash-based de-duplication system using the self detection of most repeated patterns as chunks divisors

Saja Taha Ahmed[a,*], Loay E. George[b]

[a] Ministry of Education, Vocational Education Department, Iraq
[b] Assistance of the University president for Scientific Affair, University of Information Technology & Communication, Iraq

ARTICLE INFO

Article history:
Received 22 January 2021
Revised 21 March 2021
Accepted 17 April 2021
Available online 24 April 2021

Keywords:
Big data deduplication
Hash functions
Redundancy removal
Data representation

ABSTRACT

Data reduction has gained growing emphasis due to the rapidly unsystematic increase in digital data and has become a sensible approach to big data systems. Data deduplication is a technique to optimize the storage requirements and plays a vital role to eliminate redundancy in large-scale storage. Although it is robust in finding suitable chunk-level break-points for redundancy elimination, it faces key problems of (1) low chunking performance, which causes chunking stage bottleneck, (2) a large variation in chunk-size that reduces the efficiency of deduplication, and (3) hash computing overhead. To handle these challenges, this paper proposes a technique for finding proper cut-points among chunks using a set of commonly repeated patterns (CRP), it picks out the most frequent sequences of adjacent bytes (i.e., contiguous segments of bytes) as breakpoints. Besides to scalable lightweight triple-leveled hashing function (LT-LH) is proposed, to mitigate the cost of hashing function processing and storage overhead; the number of hash levels used in the tests was three, these numbers depend on the size of data to be de-duplicated. To evaluate the performance of the proposed technique, a set of tests was conducted to analyze the dataset characteristics in order to choose the near-optimal length of bytes used as divisors to produce chunks. Besides this, the performance assessment includes determining the proper system parameter values leading to an enhanced deduplication ratio and reduces the system resources needed for data deduplication. Since the conducted results demonstrated the effectiveness of the CRP algorithm is 15 times faster than the basic sliding window (BSW) and about 10 times faster than two thresholds two divisors (TTTD). The proposed LT-LH is faster five times than Secure Hash Algorithm 1 (SHA1) and Message-Digest Algorithm 5 (MD5) with better storage saving.

1. Introduction

As data expands daily, an unprecedented increase in the amount of digital data has prompted the need to upgrade data storage devices. A study conducted by the International Data Corporation (IDC), in 2012, showed that the amount of digital information produced worldwide is approximately 1.8 ZB, and by 2020 that amount would exceed 40ZB. Moreover, almost three-quarters of digital information is duplicated (Gantz and Reinsel, 2012; Kruus et al., 2010).

According to the deduplication research made by Microsoft (Meyer and Bolosky, 2012; El-Shimi et al., 2012), and EMC (Wallace et al., 2012; Shilane et al., 2012), up to 50% and 85% of the data in their primary and secondary storage systems, respectively, are redundant and could be de-duplicated. As a result, data deduplication is defined as an efficient space and bandwidth strategy that prevents redundant data from being saved in storage devices and transferred over networks, it is one of the most effective ways to tackle this problem (Gantz and Reinsel, 2012; Kruus et al., 2010).

Deduplication removes the redundancy of data by storing one physical copy and referencing the same data to previous copied records. In general, chunk-level deduplication is more common than file-level deduplication since it recognizes and eliminates redundancy at a finer granularity. The fundamental aspect of data

* Corresponding author at: PHD from the Iraqi Commission for Computers and Informatics (ICCI), MSC in computer science from the University of Baghdad, and currently working as an instructor at the Ministry of Education.

E-mail addresses: sajataha@ymail.com, drsajataha@gmail.com (S.T. Ahmed), loaydower57@yahoo.com (L.E. George).

Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2021.04.005
1319-1578/© 2021 The Authors. Published by Elsevier B.V. on behalf of King Saud University.

deduplication is to partition the input data stream into multiple blocks (chunks). The chunking algorithms are typically categorized into two: (a) fixed-length chunking and (b) variable-length chunking (Periasamy and Latha, 2019; Sun et al., 2013).

Fixed-size chunking is the simplest and fastest, but the most important issue it is the boundary-shift problem degrades the deduplication performance; the fixed-size chunking can generate different results (i.e., different hash values) for all the subsequent chunks even though most of the data in the file are unchanged (Eshghi and Tang, 2005). On the contrary, variable-length chunking is also called Content-Defined Chunking (CDC) which proposed to address the boundary-shift problem (Zhang et al., 2015), it is declared chunk boundaries based on the byte contents of the data stream instead of on the byte offset, as in fixed-size chunking. Consequentially, most of the chunks remain unchanged when data modifications occur, thereby leading to detect further redundancy for deduplication (Moon et al., 2012; Chapuis et al., 2016) . According to some recent studies (Meister et al., 2012; Xia et al., 2016), CDC-based deduplication approaches can detect about 10–20 percent more redundancy than fixed-size chunking.

The most popular CDC approaches, like BSW and TTTD, determined chunk boundaries based on the Rabin fingerprints of the content, which is highly effective in duplicate detection but time-consuming. Since it computes and judges (against a condition value) Rabin fingerprints of the data stream byte by byte. Generally, CDC consists of two distinctive and sequential stages: (1) hashing in which fingerprints of the data contents are generated and (2) hash judgment in which fingerprints are compared against a given value to identify and declare chunk cut-points (Xia et al., 2014).

Generally, the data deduplication technique identifies and eliminates duplicated data blocks with a cryptographic hash function. Hash-based data deduplication methods use a hashing algorithm to distinguish "chunks" of data individually. The frequently used algorithms are SHA-1 and MD5. As a hashing algorithm processes data, a hash is generated that represents the data and detects the duplicate ones via certain forms of the comparison process. If the same data passes several times through the hashing algorithm, the same hash is generated each time. Deduplication, however, suffers from the long runtime and the need for more processor resources to operate on its system (Hema and Kangaiammal, 2019).

This paper introduces a deduplication system based on chunking the byte stream to find unique signatures that may be shared in multiple chunks. The contribution is new hash-based fingerprints that are applied at two stages:

1. A novel chunking approach is proposed as s commonly repeated patterns (CRP) algorithm, it is used for enhancing and simplifying the chunking approach without hash judgment by finding the most redundant divisor, to further reduce the CPU operations during CDC for data deduplication.
2. In the proposed lightweight triple-leveled hashing (LT-LH) function, each chunk has three hash values to reduce the probability of hash collision occurred during the matching stage, with 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 entire paper describes the proposed system. Some related works to data deduplication are described in section 2. The proposed framework is defined in Section 3. The outcomes of our proposed methodology are discussed in Section 4. The last section includes observations and presents some remarks for future works.

2. Related work

One of the most significant emerging technology in the field of data storage and backup research is deduplication. From different perspectives, researchers have conducted deduplication development studies, mainly based on data partitioning (feature extraction) strategies and the generation of chunk fingerprints. Several hash-based methods are suggested for fingerprint indexing, the proportion of fingerprints increased with the increase in data. The fingerprints have to be stored on disk drives because of the limited memory size. Song et al. (Song et al., 2013) introduced a fingerprint prefetching algorithm by leveraging file similarity. A similar file detection algorithm is introduced to identify similar files that are considered to have some modifications and share a large portion of the same data blocks. The suggested algorithm prepared fingerprints from disk drives and placed them in memory before they are needed. This will significantly improve the cache hit ratio when fingerprints are needed, but the fingerprint generation was not adopted. Moreover, the experiments conducted using chunks size vary from 4 K to 8 K, and 16 K.

Since chunks of existing deduplication range in size from 4 KB to over 16 KB, existing systems do not apply to the datasets consisting of short records. a new framework presented by Hou et al. (2017) designed for short fingerprints (SH-Dedup), inline, hash-collisions-resolved deduplication system, is capable of applying the deduplication process on a large set of mobile Internet data, each chunk may be smaller than 100 B or even smaller than 10 B in size. Experimental application findings showed that SH-Dedup is capable of reducing the storage space but is specific to short text strings. Angelina et al. (2018) presented a novel mechanism to reduce hash collisions during insertions in a hash indexing structure using five hash functions. The Permutation Hashing is based on chunks of 10 K; 100 K, 1 M, and 10 M. Permutations are performed selectively over the fingerprint if a collision occurs to determine the new index location in the hash structure. With this strategy, the lookup latency involved in the De-duplication method is reduced, but the deduplication ratio and throughput were neglected.

Traditional CDC-based approaches were introduced with high CPU overhead because they mark the chunk cut-points by computing and judging the rolling hashes of the data stream byte by byte. Xia et al. (2020) proposed the FastCDC approach by making combined use of five key techniques, namely, gear based fast-rolling hash, simplifying and enhancing the Gear hash judgment, skipping sub-minimum chunk cut-points, normalizing the chunk-size distribution in a small specified region to address the problem rolling two bytes each time to further speed up CDC. The evaluation results show that there was an enhancement in deduplication throughput and consuming time while achieving nearly the same and even higher deduplication ratio as the classic Rabin-based CDC.

3. The proposed system

The proposed system's input data covers a range of files of various sizes and types. These files are processed by the system as one file at a time. The proposed system was developed and tested on three dataset versions belonging to the Linux file system (Linux-3.9, Linux-4.14.157, Linux-5.8.12), these versions consisting of 173,109 files, and 11,705 folders of 2,32 GB to demonstrate the efficiency of the proposed system. Any deduplication system can usually pass into three main phases: (a) chunking, (b) hashing and indexing, and (c) matching phases

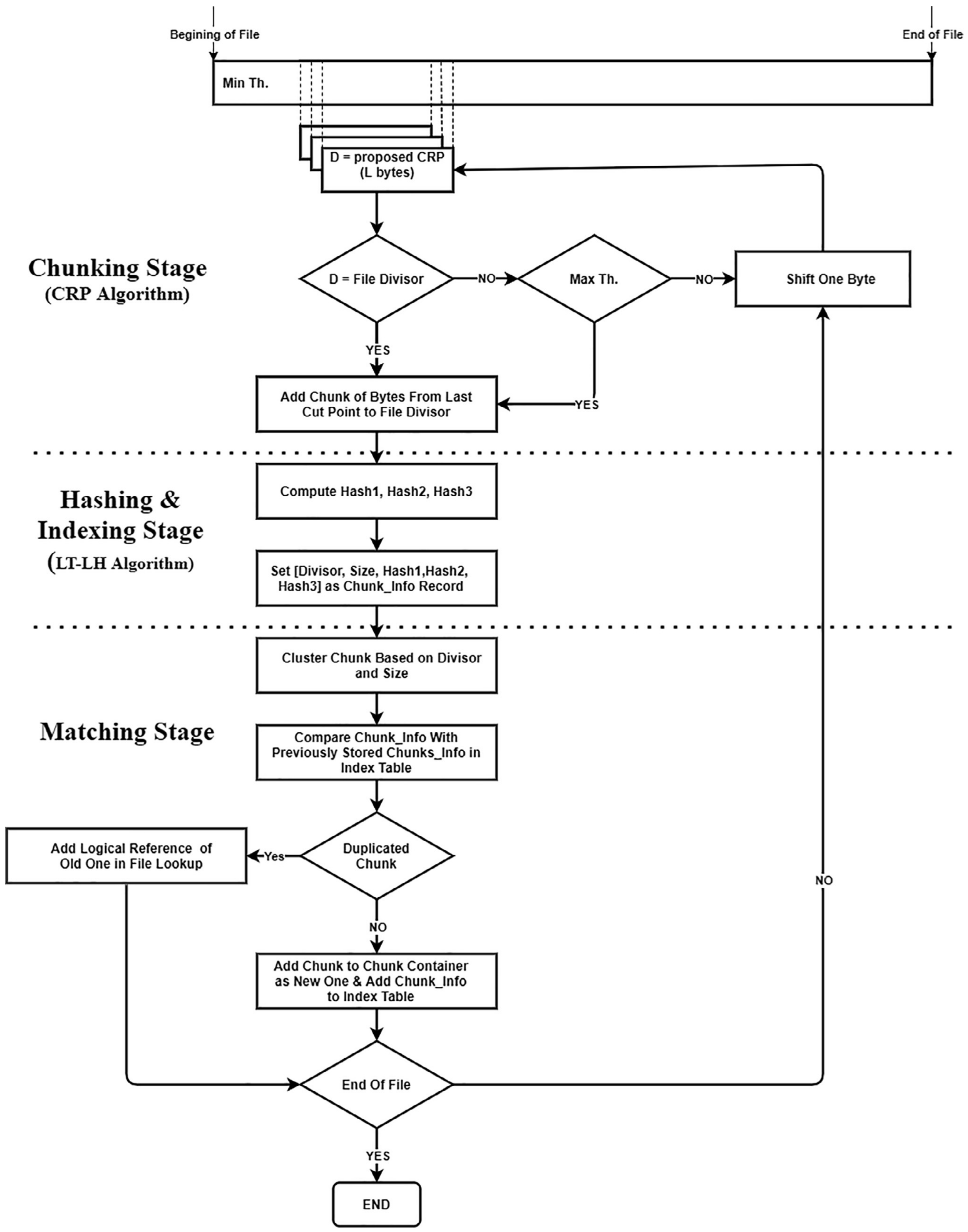
Begining of File
End of File
Min Th.
D = proposed CRP (L bytes)
Chunking Stage (CRP Algorithm)
D = File Divisor
NO
Max Th.
NO
Shift One Byte
YES
YES
Add Chunk of Bytes From Last Cut Point to File Divisor
Hashing & Indexing Stage (LT-LH Algorithm)
Compute Hash1, Hash2, Hash3
Set [Divisor, Size, Hash1,Hash2, Hash3] as Chunk_Info Record
Cluster Chunk Based on Divisor and Size
Matching Stage
Compare Chunk_Info With Previously Stored Chunks_Info in Index Table
Duplicated Chunk
Yes
Add Logical Reference of Old One in File Lookup
NO
Add Chunk to Chunk Container as New One & Add Chunk_Info to Index Table
End Of File
NO
YES
END

Fig. 1. The proposed Deduplication Stages for Single File.

Fig. 1 shows the proposed deduplication system processes. Every stage will be clarified in terms of the proposed algorithm in the following subsections:

3.1. The proposed CRP chunking algorithm

Chunking is the first stage in the deduplication system. It is based on a certain form of rolling hash, partitioning the input data stream (file) into small and non-overlapping segments. A new CRP algorithm scans the file to find out the most probable hash value for the sequence of bytes found in the dataset, to be selected as a divisor.

The proposed chunking algorithm is based on two main steps:

1. Compute the hash value for each byte sequence of length L.
2. Finding the most occurrence hash values to be selected as a set of divisors in the file.

The key concept is that there are terms that have very low discrimination value in file chunking, this implies that these phrases hold unimportant duplicated knowledge that can't be a proper signature for deduplication. Thus, the recommended deduplication system is worth ignoring all these terms in the process of file chunking. The proposed CRP algorithm avoids these words based on the most founded patterns in the file with the ability to determine the length of the bytes stream as breakpoints or divisors for that file. In other words, some internal features of the files are represented as cut points. The proposed CRP uses the highest probable hash of these bytes included in the chunk and situated at the boundary of the chunk. The nominated hash is exploited to specify the cut points instead of using judges according to a condition value at each rolling hash computation (i.e., traditional Rabin fingerprint), which required more processing time.

In the proposed CRP fingerprinting function for each L sequence of characters, the fingerprint value is calculated sequentially using Eqs. (1)–(3). This allows the proper divisor to be identified, as follows:

$$D_1 = \sum_{i=0}^{l} (A_1[i] * S[i]) \ \& \ 0x7FFF \tag{1}$$

$$D_2 = \sum_{i=0}^{l} (A_2[i] * S[i]) \ \& \ 0X7FFF \tag{2}$$

$$Divisor = D_1 \ or \ (D_2 \ll 15) \tag{3}$$

where, L represents the length of the divisor for each S sequence of byte(s), it was set either 5, or 6,7,8, or 9 bytes, A_1 & A_2 are two arrays of random number generated, 0x7fff hex (=$2^{15}-1$) used to get a limited range of value. The CRP fingerprint can be illustrated in Fig. 2.

The proposed algorithm employs a hash function to indicate the CDC-based variable size that limits the effect of adding or deleting characters in the particular region (i.e. avoiding the boundary shifting problem). The proposed algorithm uses Rolling Hash and a predefined sliding window to flow through the file's full stream of bytes. A sliding window will be rolling one byte removing the first byte of the L bytes, append the new one, and send the new block to the CRP algorithm to calculate its fingerprint value (i.e., divisor value), then hold all computed hash values along with their frequencies in a hash table. For that file, the most redundant hash will be recognized as the divider and will be used to define the position of the breakpoint. Consequently, if this divisor is replicated at any place, it will again be treated as a breakpoint and that will lead to the fastest possible nomination of a duplicate chunk in a big or different file. Since it determines the divisor in the file without relying on any conditions, as in the conventional chunking algorithm.

Moreover, minimum and maximum thresholds chunking conditions are often used as a hard threshold to avoid chunks being with a very small size or too big sizes. These two factors reduce the variations of the chunk sizes. The proposed chunking algorithm based on CRP fingerprint is shown in Fig. 3 and the detailed steps are explained in Algorithm 1.

Algorithm 1 The CRP Chunking Algorithm

```
Objective: Divide the file into chunks based on File Divisor,
  Min Th., and Max Th.
Input: File of different sizes and types as an array of bytes.
  Min Th., Max Th. and File Divisor computed as in Fig. 1.
Output: Chunks of different sizes.
Step 1: Set Len ← Length of File
   Set L ← Divisor Length
   Set Startpoint ← 0
   Set Endpoint ← 0Step 2: While Not End of File Do
   If Len = 0 Goto Step 5
   ElseIf Len < = Min Th.
      Set Chunk ← Bytes of File from Startpoint to (Startpoint + Len), Goto Step 5
   ElseIf Len <= (Min Th. + L)
      Set Chunk ← Bytes of File from Startpoint to (Startpoint + Len + L), Goto Step 5
   Else
      Endpoint ←(Startpoint + Min Th.-1)
      Set window ← Bytes of File from Endpoint To Endpoint + L
   Step 3: Compute Divisor (window) using Eq. (3).
   If Divisor (window) = File Divisor
      Set Chunk ← Bytes of File from Startpoint To Endpoint + L
      Startpoint ← Endpoint + 1
      Len ← Len- Startpoint, Go to Step 2
   Else Go to Step 4
   End F
   Step 4: If (Endpoint- Startpoint) >= Max Th.
   Set Chunk ← Bytes of File from Startpoint to Startpoint + Max Th.
   Set Startpoint ← Endpoint + 1
   Len = Len – Startpoint: Goto Step 2
      Else
   Shift window one byte
   Set Endpoint ← Endpoint + 1
   Go to Step 3Step 5: End
```

3.2. The proposed LT-LH algorithm

The traditional deduplication system is suffering from wasted time and storage needed to solve the collision problem. An LT-LH function is suggested, which is based on a linear bound sum of a string of non-repeatable zero bytes multiplied by a random number sequence to construct a hash value for the sequence representation. Moreover, using more than one hash sequence to reflect the content of the string will generate a combined hash value that has smaller collision hit rates. Thus, for each chunk, the technique can compute and save three hash values using the hash functions shown below:

$$Hash_1(chunk) = \sum_{i=0}^{k} (S[i] * H_1[(i \ mod \ 255) + 1]) \ \& \ 0xFFFF \tag{4}$$

$$Hash_2(chunk) = \sum_{i=0}^{k} (S[i] * H_2[(i \ mod \ 255) + 1]) \ \& \ 0xFFFF \tag{5}$$

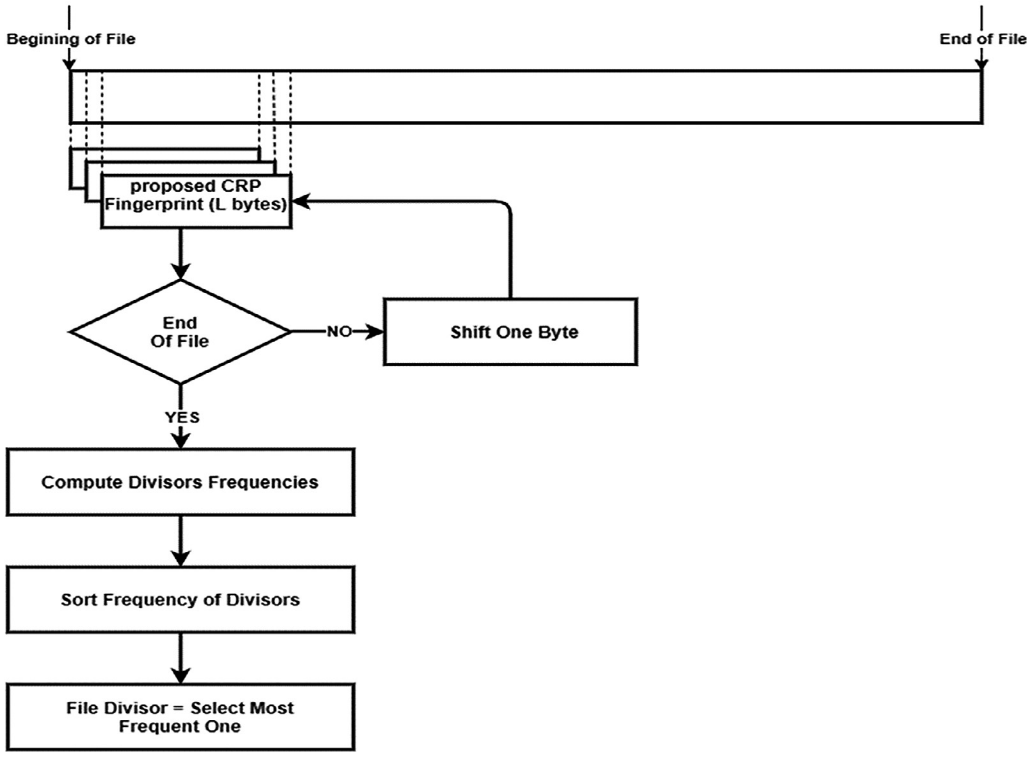
Begining of File
End of File
proposed CRP Fingerprint (L bytes)
End Of File
NO
Shift One Byte
YES
Compute Divisors Frequencies
Sort Frequency of Divisors
File Divisor = Select Most Frequent One

Fig. 2. The Proposed File Divisor (CRP Fingerprint) Layout.

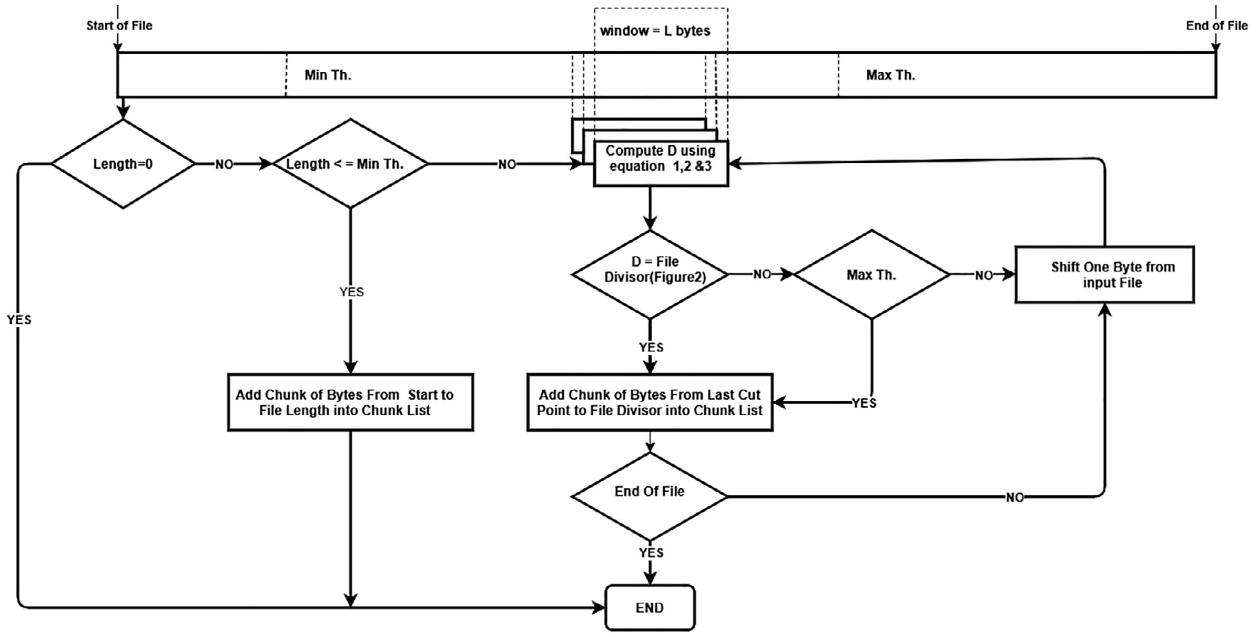
Start of File
window = L bytes
End of File
Min Th.
Max Th.
Length=0
NO
Length < = Min Th.
NO
Compute D using equation 1,2 &3
YES
YES
D = File Divisor(Figure2)
NO
Max Th.
NO
Shift One Byte from input File
YES
Add Chunk of Bytes From Start to File Length into Chunk List
Add Chunk of Bytes From Last Cut Point to File Divisor into Chunk List
YES
End Of File
NO
YES
END

Fig. 3. The Proposed CRP Chunking Algorithm.

$$Hash_3(chunk) = \sum_{i=0}^{k}(S[i] * H_3[(i \bmod 255) + 1]) \ \& \ 0xFFFF \qquad (6)$$

where S is an array of chunk bytes, k is the length of a chunk, 0xFFFF hex (=2^{16} in decimal) is used to get a limited range of values.0

H_1, H_2 & H_3 are three arrays of 255 generated random numbers. The generated arrays use any fixed sequence of numbers via exploiting the rand () function for generating a sequence of numbers for one time and save these sequences to be used in Eqs. (4)–(6), respectively. Procedure 1 is used to generate a random number array simply.

Procedure 1 The Rand Array Generator Algorithm

```
Objective: generate three arrays of 255 random number
Output: an array of random numbers.
For I = 0 To 255: H1(I) = I: next I
For I = 255 To 1 step −1: K = rand(I): Swap H1(I), H1(K): Next I
For I = 0 To 255: H2(I) = I: next I
For I = 255 To 1 step −1: K = rand(I): Swap H2(I), H2(K): Next I
For I = 0 To 255: H3(I) = I: next I
For I = 255 To 1 step −1: K = rand(I): Swap H3(I), H3(K): Next I
```

The use of a different sequence of random numbers to produce many short hash values is adequate to produce different signatures to characterize the plain text contents of chunks. Moreover, the mathematical operation that used to compute the hash values are simple; only primitive mathematical operations are used, which in turn make the required cost of computing the hash too low in comparison with the traditional security functions, besides their length (in bits) is bounded and does not depend on the length of the big-data size and characteristics.

The use of long hashes decreases the probability of hash collision, but requires more processing time and raising the overhead information that is required in the matching stage, so the selection of hash function is greatly application accuracy dependent. Usually, the two most prominent functions are MD5 and SHA-1, which are commonly used in deduplication. Generally, the probability of hash collisions depends on the strength of the hashing algorithm. Since hashing is CPU intensive, so the use of suggested (LT-LH) functions reduces the matching time by effectively solving the collision problem, the number of bits required to store these three hashes is packed into 48 bits maximum (equals 12 hexadecimal digits), which is less than the number of bits required to save the hash value in SHA-1 and MD5, which yields 160 bits for SHA-1 signature (i.e., equals 40 hexadecimal characters); and for MD5 hash system, the produced hash is 128 bits, which equals to 32 hexdecimal characters.

In the proposed deduplication system, the chunk container incorporates unique chunks, so each chunk has an identifier that represents the chunk index in the chunk container and takes an integer number from zero to N; where N is increased limitlessly for each chunk generated in the system. For each chunk, the name, the divisor, the size, and the three hash values must be saved in the Index-Table that can be explained in Table 1.

3.3. The proposed matching process

When a new file is entered, it passes the preceding two phases, the system must define and remove the duplicated chunks in the deduplication matching steps. Firstly, the traditional algorithm compares the hash values of the chunks, if they are the same, then the algorithm will compare the two chunks byte to byte, if they are identical the system deletes the new one and adds a logical reference to the location of the old one. Otherwise, there is a collision incidence; the chunks are different; the new one would be saved as a new chunk by the system. This operation requires a lot of time and overhead the system. Therefore, a new approach will be adapted to the deduplication matching process to improve the throughput, i.e. to save execution time and reduce the usage of Processor resources.

The index table information is suggested in this paper to optimize the matching process using the divisor, the size, and the three hash values that have already been determined in the hashing step. For each chunk, this data acts as a descriptor, the chunks cluster according to their divisor hash value, and it further groups within the same category of unique divisors according to their size. Finally, three hashes are checked, if a collision occurs in the first hash values of the compared chunks, then the second and third hashes are compared. The test result shows a significantly notable improvement with the time taken by the matching process because it is faster to group the chunks concerning their divisors and sizes, then compare three hashes than to compare the whole of the comparable chunks byte-byte.

4. Results and discussion

The suggested system was realized on a computer that has the following hardware equipment: Intel CORE-i7 CPU configuration with installed memory of 12.00 GB on a 64-bit Windows operating system. Besides that, the theoretical methodology implemented using C# and Visual Studio 2015. The test results are performed using the following performance metrics to perform accurate and equitable assessments:

- Deduplication Ratio (DR): The data deduplication ratio measures the effectiveness of the deduplication process; it is expressed using Eq. (7) (Fu et al., 2011):

$$DR = \frac{Total\ Input\ Data\ Size\ Before\ Deduplication}{Total\ Input\ Data\ Size\ After\ Deduplication} \quad (7)$$

- Average Chunk size: can be calculated by dividing the total size of the input data by the total number of chunks (Romański et al., 2011).
- Chunking and Hashing time: It is the total time taken to perform hashing and chunking operations (Yoon, 2019).

The tests are performed following viewpoints; the first one is studying and analyzing the behavior of the proposed system concerning the length of the divisor (i.e., CRP fingerprint length). While the second, compare the system behavior after determining the optimal parameters with a conventional method to illustrate the proposed deduplication system efficiency.

4.1. Choosing the optimum CRP fingerprint length

This section presents the examination of some CRP algorithm's properties, mainly the influences of CRP fingerprint length on the deduplication ratio, average chunk size, and chunking time.

4.1.1. The influence of CRP length on deduplication ratio

In general, the chunking algorithm has several significant impacts on the performance of the data de-duplication systems. To determine shift boundary issues, the proposed deduplication model establishes the chunk boundaries by the content of a file.

Table 1
The Index Table Structure.

Chunk Name	Divisor Hash	Chunk Size (Bytes)	Hash 1	Hash 2	Hash 3
0	16,320	627	55,434	57,737	21,156
1	20,348	159	54,678	35,617	33,431
2	28,590	512	12,010	11,164	45,946

Thus, the proposed CRP fingerprint length has a noteworthy influence on chunking approach fulfillment when tending to exploit the most similar breakpoints among various files in the tested dataset. Table 2 shows the proposed system results in terms of the performance metrics. The tested length from 3 to 10 bytes with a minimum chunk size of 128 and a maximum chunk size of 1024 bytes, the last two parameters considered fixed to demonstrate the effectiveness of proposed CRP fingerprint length on performance criteria.

We can observe that the system can attain a higher deduplication ratio when the divisor gets a shorter content segment, choosing a long breakpoints segment on the other side may not contribute to duplicated chunks being discovered. Fig. 4 illustrated the effect of divisor length on the deduplication ratio. Since related chunks that appear in the same file or even in multiple files shared repeated patterns represented unique signatures, therefore, increasing cut-point section leads to remove the commonly shared patterns across files on that volume. The maximum deduplication performance is achieved for a CRP length of three bytes, where the deduplication ratio is 3.11.

4.1.2. CRP length effectiveness on average chunk size

The number of chunks will increase if a shorter divisor is considered as presented in Table 2. For the hash-based deduplication systems, the increasing number of chunks often means increasing the size of the lookup table, and so the systems need to invest more time to perform the comparisons. But the proposed CRP fingerprint technique used in the chunking stage increases the number of small chunk sizes based on shorter length divisor, with less CPU overhead cost, which in turn increases the deduplication ratio with enhancement in chunking time. Improving DR implies eliminating more duplicated chunks, which can reduce the lookup table size. Fig. 5 illustrates the relation between CRP divisor length and the average size of the chunks. So that CRP fingerprint of length 3 bytes has less average chunking size of 209 bytes.

4.1.3. The effect of CRP length on CRP Divisor, Maximum, and minimum thresholds chunking ratio

The maximum and minimum criteria are used to exclude very large and very small-sized chunks for the management of chunk-size variability. The CRP divisor plays a vital role to make the chunk-size close to the expected chunk-size. In other words, the CRP divisor prevents the algorithm from relying on the maximum threshold as the breakpoint and splits the data stream using its internal feature for better performance of the deduplication system. In this paper, we compare the CRP divisor, maximum, and minimum threshold chunking operation. The result is presented in Table 3.

The inferred truth is the CRP-based chunking of 3 bytes' length obtains the highest number of 11,729,570 chunks which reflecting a ratio of 98.6% over the other two thresholds. As long as the length of CRP getting longer, the maximum threshold will have a higher

Table 2
The Effect of CRP Length on Deduplication System.

CRP length	Duplicated Chunk No.	De-duplicated Chunk No.	Deduplication Ratio
3	11,896,717	4,071,291	3.11
4	8,609,615	2,881,301	2.76
5	7,943,018	2,666,045	2.67
6	7,435,691	2,514,863	2.59
7	7,058,912	2,403,952	2.52
8	6,683,511	2,314,003	2.41
9	6,348,287	2,216,999	2.35
10	5,985,413	2,144,939	2.28

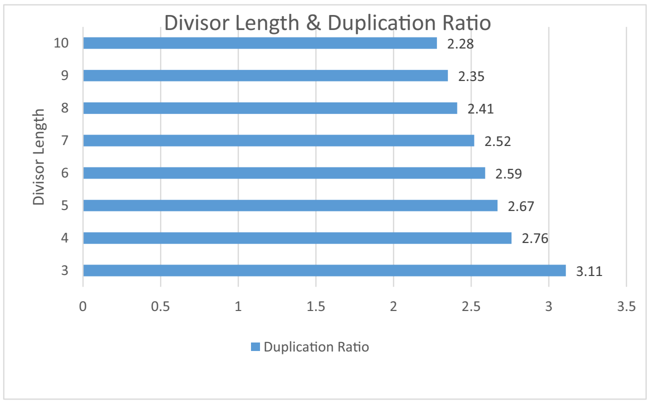

Fig. 4. CRP Length Effectiveness on Deduplication Ratio.

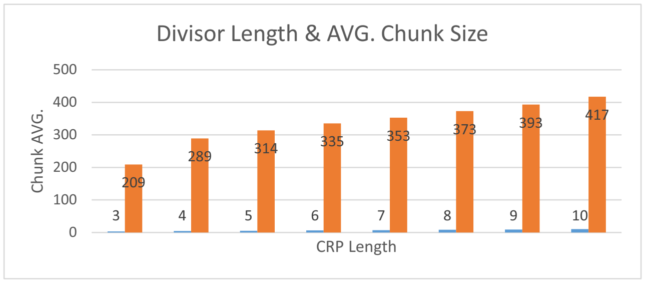

Fig. 5. CRP Length Effectiveness on Average Chunk Size.

number of chunks and the CRP divisor will get decreased effects on the chunking process. On the other side, the minimum threshold has the lowest influence in the chunking stage. The CRP divisor has a dominant behavior in the deduplication system that reflects the importance of utilizing internal characteristics of datasets and regulating variation of chunks size.

4.2. Different chunk size impact

The deduplication ratio is directly influenced by chunk size, to make a scientific experiment more accurate, the various chunk size is tested using different Min-Max thresholds. More duplicated chunks are identified by small Min-Max Th. values, but they influence the size of metadata. The deduplication ratio is decreased by high Min-Max Th. values because they create large fragments, which further decreases redundant data detection in turn. Table 4 indicates the dataset size after deduplication.

As explained earlier the optimal CRP length was set to 3 bytes, the results show that the storage size after duplication elimination is optimal by setting the chunk size between 128 and 1024 bytes. Deduplication at that chunk size reduced data from 2.32 GB to 802 MB. The findings indicate chunk size plays an important role in deduplication, given the major benefit shown by small block sizes.

4.3. The proposed CRP chunking algorithm efficiency

The basic concept from previous experimentation has shown the optimal chunking parameters, the traditional chunking algorithm is a time- and resource-consuming operation to locate the breakpoints since it must search the full file byte by byte. Also, the chunking stage is entirely dependent on the conditions of the file-breaking chunking algorithms. In this paper, the CRP is compared with BSW and TTTD algorithms. In CRP, to discover full

Table 3
CRP Divisor, Max Th., Min Th. Comparison.

CRP length	Duplicated Chunk No.	Chunking based on CRP		Chunking based on Max.		Chunking based on Min.	
		#Chunk	Ratio	#Chunk	Ratio	#Chunk	Ratio
3	11,896,717	11,729,570	98.6%	162,837	1.36%	4310	0.036%
4	8,609,615	8,154,104	94.7%	451,132	5.24%	4379	0.051%
5	7,943,018	7,401,914	93.19%	536,692	6.76%	4412	0.056%
6	7,435,691	6,817,930	91.7%	613,317	8.25%	4444	0.059%
7	7,058,912	6,379,767	90.37%	674,657	9.55%	4488	0.063%
8	6,683,511	5,917,505	88.50%	761,467	11.39%	4539	0.067%
9	6,348,287	5,507,284	86.75%	836,414	13.17%	4589	0.072%
10	5,985,413	5,073,938	84.77%	906,853	25.15%	4622	0.077%

Table 4
Different Chunk Size Based on Optimal CRP of Three Bytes Length.

Chunk Size	Duplicated Chunk No.	Deduplicated Chunk No.	Dataset Size after Deduplication (MB)	DR
128–256	11,752,609	4,680,139	915	2.72
128–512	10,175,143	3,584,837	864	2.88
128–1024	11,896,717	4,071,291	802	3.11
128–2048	9,364,853	3,084,832	875	2.85
128–4096	9,296,166	3,040,368	883	2.82
128–8192	9,276,320	3,026,752	887	2.81
256–512	2,893,784	6,338,208	1030	2.42
256–1024	3,382,457	1,790,330	1168	2.13
256–2048	5,546,617	2,265,840	985	2.53
256–4096	5,479,973	2,218,775	989	2.52
256–8192	5,460,582	2,204,682	993	2.51
512–1024	3,382,457	1,790,330	1129	2.20
512–2048	3,184,142	1,598,610	1131	2.20
512–4096	3,121,127	1,547,711	1129	2.20
512–8192	3,102,579	1,532,788	1131	2.20

redundancy, the system will dynamically determine the divisor based on the statistical analysis of file content and therefore will identify the duplicated chunks quickly than other strategies. The conducted results show that CRP is faster than BSW about 15 times and 10 times faster than TTTD. In terms of deduplication ratio, the CRP outperformed the two algorithms by 3.11, represents 4,071,291 de-duplicated chunks (without redundancy) out of all 11,896,717 chunks, as shown in Table 5.

4.4. The proposed lightweight triple-level hashing function

The proposed LT-LH function used in the hash-based deduplication system to generate the chunk's fingerprint has a direct impact on the size of hashes stored in the index table. Moreover, the proposed hash function is built on choosing random quantities retrieved from a random number generated array, the aim of generating random numbers for the hashing function is to obtain a unique hash value for a given chunk. Besides, the successive numbers produced by the random number generator are independent of a given distribution.

Each proposed fingerprint requires 48 bits (6 bytes), while the standard MD5 and SHA1 hash functions require 128 bits (16 bytes) and 160 bits (20 bytes), respectively. The impact of the LT-LH algorithm on the storage size needed to store fingerprints in the index table computed by Eq. (8) is shown in Fig. 6. Concerning that, the hashes produced are based on optimal parameters extracted in previous experiments, 128–1024 bytes of chunk size limits and three bytes of CRP length. Besides, the total number of chunks after deduplication was 4,071,291 for the tested Dataset.

$$\text{Hashing index table size} = \text{No. of Bytes per Fingerprint} \times \text{Total No. of Chunks after DD}, \quad (8)$$

Moreover, the resulted hash value is used to denote chunks using the linear bounded sum of a string of non-repeatable zero bytes multiplied by a random sequence of numbers. The proposed LT-LH function is based on a simple mathematical operation compared to the standard hashing functions (SHA1 and MD5), which consume significant processing resources and contribute to high

Table 5
Chunking Time and DR of CRP, BSW, and TTTD.

	Chunking Time (Sec.)	Chunking Throughput (MB/Sec)	Deduplication Ratio
CRP	62	40.28	3.11
BSW	957	2.61	1.7
TTTD	633	3.94	2.01

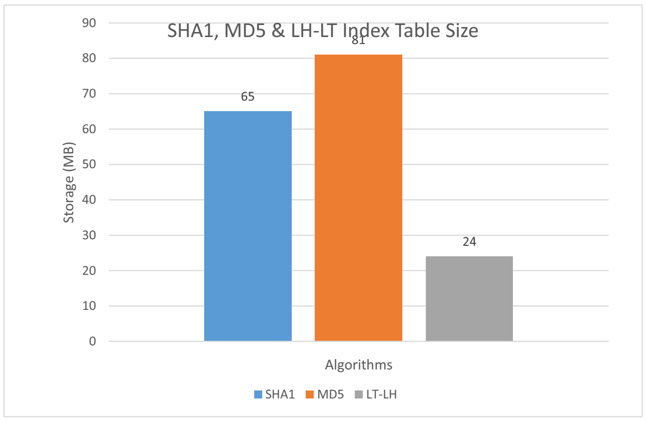

Fig. 6. The Index Table Size of SHA1, MD5, and LT-LH.

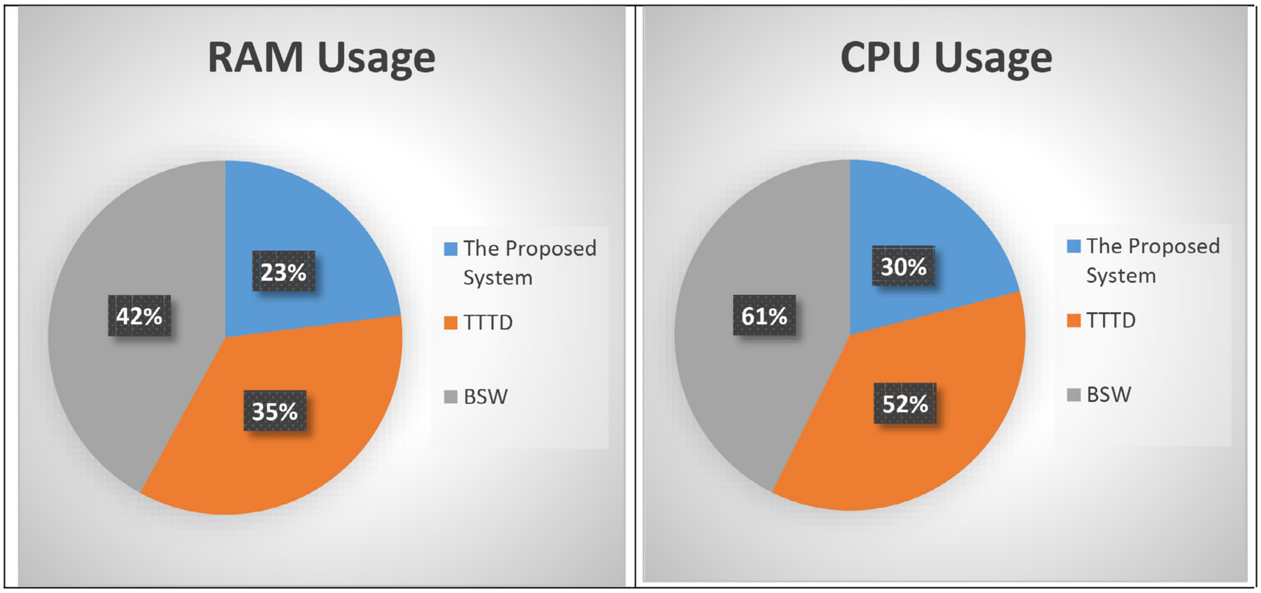

Fig. 7. The Resources Usage Comparison.

Processor overhead when calculating hash values. The effect of hash algorithms on hashing stage time and throughput, determined by Eq. (9) and shown in Table 6.

$$\begin{aligned} \textit{Hashing Algorithm Throughput} &= \textit{Hashing Computational overhead} \\ &= \frac{\textit{Processed Data in MB}}{\textit{Time in Second}} \end{aligned} \tag{9}$$

4.5. The resources usage

The proposed deduplication system used an optimized matching process based on index table information instead of byte-byte comparison. Moreover, it proposed some lightweight operations to save the storage space and reduce processor overhead, so it can be hand over resources to some other applications. Visual Studio .NET provides a performance profile to measure memory and CPU usage while an application is running. The amount of RAM utilized by the proposed system is about 2.15 GB of a total of 12 GB, which is represented about 23%. Besides, the total processor usage is about 30%. Fig. 7 shows the percentage of utilized RAM and processor between our system and traditional ones which are consuming more resources.

4.6. The proposed system in comparison with other work

The performance of the proposed approach is compared with the performances of some published deduplication approaches, the findings revealed that the superiority of the proposed CRP and LT-LH approaches in their ability to distinguish more duplicated chunks as easily as possible over state-of-the-art deduplication approaches. Luo et al. (2015) proposed an approach of deduplication that used a routing algorithm for a web dataset of 52 GB, even it used a larger dataset but it can reach a deduplication ratio of 2.25 in comparison with our DR of 3.11 for 2.32 GB Linux dataset. In terms of chunking time, Jasim and Fahad (2018) suggested a new deduplication system to solve collision problems using two 580 MB and 1.27 GB datasets, respectively. The chunking time is 462 s and 1075 s for two datasets, respectively. This demonstrated the importance of our suggested system of chunking the 2.32 GB dataset in 62 s.

Table 6
SHA1, MD5, and LT-LH Hash Time.

	Hash Time (Sec.)	Hashing Throughput (MB/s)
SHA1	87	28.70
MD5	79	31.60
LT-LH	15	166

5. Conclusions

The analysis of the relationship and effect of different parameters on the efficiency of the deduplication system using new CRP chunking and LT-LH hashing algorithms is considered in this study. By reducing space consumption and maximizing CPU throughput, the composition of two new algorithms offers remarkable storage efficiency. The CRP chunking algorithm is suggested to enhance the deduplication ratio and chunking time that leads to improving chunking throughput. For reducing the time of the hashing phase, the LT-LH is proposed to save the storage space of the index table and increase hashing throughput.

The conducted results showed that the triple group of bytes has higher DR and consuming less time for chunking process fulfillment. Also, the optimum Min-Max Th. is 128–1024 bytes, these variables affect the CRP chunking algorithm to be faster than BSW about 15 times and 10 times faster than TTTD. Besides, the experiments showed that for the index table, the proposed LT-LH occupies 24 MB compared to SHA1 and MD5 that take 65 and 81 MB, respectively. The proposed function implies simple mathematical operations along with less computation time of about 15 s than the conventional hashing algorithm.

In future work, we will study dynamic allocation for the used set of divisors, i.e. using a set of divisors updatable with time using monitoring tools for the incoming streams of bytes, and using file categorization or classifier to use different divisors and hash functions. Moreover, we will investigate bigger datasets when the computer configuration is upgraded.

Finally, a better tradeoff between the influencer parameters on the deduplication system regarding high duplication detection with less overhead computation must be accomplished by the successful deduplication system.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Gantz, J., Reinsel, D., 2012. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. IDC iView: IDC Analyze the Future 2007 (2012), 1–16.

Kruus, E., Ungureanu, C., Dubnicki, C., 2010. Bimodal content defined chunking for backup streams. In: Fast, pp. 239–252.

Meyer, D.T., Bolosky, W.J., 2012. A study of practical deduplication. ACM Trans. Storage (ToS) 7 (4), 1–20.

El-Shimi, A., Kalach, R., Kumar, A., Ottean, A., Li, J., Sengupta, S., 2012. Primary data deduplication—large scale study and system design. In: Presented as part of the 2012 {USENIX} Annual Technical Conference ({USENIX}{ATC} 12), pp. 285–296.

Wallace, G., Douglis, F., Qian, H., Shilane, P., Smaldone, S., Chamness, M., Hsu, W., 2012. Characteristics of backup workloads in production systems. In: FAST, p. 4.

Shilane, P., Huang, M., Wallace, G., Hsu, W., 2012. WAN-optimized replication of backup datasets using stream-informed delta compression. ACM Trans. Storage (ToS) 8 (4), 1–26.

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

Sun, Z., Shen, J., Yong, J., 2013. "A novel approach to data deduplication over the engineering-oriented cloud systems. Integr. Comput. -Aided Eng. 20 (1), 45–57.

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

Zhang, Y., Jiang, H., Feng, D., Xia, W., Fu, M., Huang, F., Zhou, Y.A.E., 2015. An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication. In: 2015 IEEE Conference on Computer Communications (INFOCOM). IEEE, pp. 1337–1345.

Moon, Y.C., Jung, H.M., Yoo, C., Ko, Y.W., 2012. Data deduplication using dynamic chunking algorithm. In: International Conference on Computational Collective Intelligence. Springer, Berlin, Heidelberg, pp. 59–68.

Chapuis, B., Garbinato, B., Andritsos, P., 2016. Throughput: A key performance measure of content-defined chunking algorithms. In: 2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW). IEEE, pp. 7–12.

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

Xia, W., Zhou, Y., Jiang, H., Feng, D., Hua, Y., Hu, Y., Zhang, Y., 2016. Fastcdc: a fast and efficient content-defined chunking approach for data deduplication. In: 2016 {USENIX} Annual Technical Conference ({USENIX}{ATC} 16), pp. 101–114.

Xia, W., Jiang, H., Feng, D., Tian, L., Fu, M., Zhou, Y., 2014. Ddelta: a deduplication-inspired fast delta compression approach. Perform. Eval. 79, 258–272.

Hema, S., Kangaiammal, A., 2019. Distributed storage hash algorithm (DSHA) for file-based deduplication in cloud computing. In: International Conference on Computer Networks and Inventive Communication Technologies. Springer, Cham, pp. 572–581.

Song, L., Deng, Y., Xie, J., 2013. Exploiting fingerprint prefetching to improve the performance of data deduplication. In: 2013 IEEE 10th International Conference on High-Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing. IEEE, pp. 849–856.

Hou, Z., Chen, X., Wang, Y., 2017. Content-level deduplication on mobile internet datasets No. 1, p. 020086. AIP Conference Proceedings. AIP Publishing LLC.

Angelina, J., Rubel, Jane, Sundarakantham, K., 2018. Collision Handling in a Data Deduplication System Using Permutation Hashing.

Xia, W., Zou, X., Jiang, H., Zhou, Y., Liu, C., Feng, D., Zhang, Y., 2020. The design of fast content-defined chunking for data deduplication based storage systems. IEEE Trans. Parallel Distrib. Syst. 31 (9), 2017–2031.

Fu, Y., Jiang, H., Xiao, N., Tian, L., Liu, F., 2011. Aa-dedupe: An application-aware source deduplication approach for cloud backup services in the personal computing environment. In: 2011 IEEE International Conference on Cluster Computing. IEEE, pp. 112–120.

Romański, B., Heldt, Ł., Kilian, W., Lichota, K., Dubnicki, C., 2011. Anchor-driven subchunk deduplication. In: Proceedings of the 4th Annual International Conference on Systems and Storage, pp. 1–13.

Yoon, M., 2019. A constant-time chunking algorithm for packet-level deduplication. ICT Express 5 (2), 131–135.

Luo, S., Zhang, G., Wu, C., Khan, S., Li, K., 2015. Boafft: distributed deduplication for big data storage in the cloud. IEEE Trans. Cloud Comput.

Jasim, H.A., Fahad, A.A., 2018. New Techniques to Enhance Data Deduplication using content-based-TTTD Chunking Algorithm. Int. J. Adv. Comput. Sci. Appl. 9 (5), 116–121.