THE USE OF ROUGH CLASSIFICATION AND TWO THRESHOLD TWO DIVISORS FOR DEDUPLICATION

Hashem B. Jehlol [1]
Iraqi Commission for Computers and Informatics

Informatics Institute of Postgraduate Studies
Baghdad-Iraq
phd202020556@iips.icci.edu.iq

Loay E. George
University of Information Technology and Communication (UoITC)
Baghdad-Iraq

loayedwar57@uoitc.edu.iq

Abstract **- The data deduplication technique efficiently reduces and removes redundant data in big data storage systems. The main issue is that the data deduplication requires expensive computational effort to remove duplicate data due to the vast size of big data. The paper attempts to reduce the time and computation required for data deduplication stages. The chunking and hashing stage often requires a lot of calculations and time. This paper initially proposes an efficient new method to exploit the parallel processing of deduplication systems with the best performance. The proposed system is designed to use multicore computing efficiently. First, The proposed method removes redundant data by making a rough classification for the input into several classes using the histogram similarity and k-mean algorithm. Next, a new method for calculating the divisor list for each class was introduced to improve the chunking method and increase the data deduplication ratio. Finally, the performance of the proposed method was evaluated using three datasets as test examples. The proposed method proves that data deduplication based on classes and a multicore processor is much faster than a single-core processor. Moreover, the experimental results showed that the proposed method significantly improved the performance of Two Threshold Two Divisors (TTTD) and Basic Sliding Window BSW algorithms..**

Index Terms - Big Data, Deduplication, hash function, data classification, Multicore processing .

I. INTRODUCTION

In recent years, there has been an astronomical increase in the volume of digital data[1]. Increased data collection and storage need more computing power, storage space, and network bandwidth [2]. The cost-effectiveness of storage management is a significant challenge [3]. The big data era has a fast growth rate, a large variety of data types and sources, and low-value density [4]. Google, International Business Machines Corporation IBM, Microsoft, Intel, and Motorola reported that 75 per cent of big data is duplicate [5]. Deduplication effectively reduces redundant data, as it avoids storing and transmitting duplicate data across networks. Storage systems widely use data deduplication to improve storage efficiency and input/output performance [6].

Data deduplication often uses Content Defined Chunk (CDC) due to its high potential and ability to find and remove duplicate data in different systems [7]. However, implementing CDC adds considerable performance overhead to the system, mainly when variable-chunking size is employed, commonly used in backup storage systems [8]. In addition, the chunking, hashing, and matching for deduplication are time-consuming and CPU intensive. For example, calculating the hash value from a Rapin-based rolling window splitting per byte of file offset uses a significant amount of CPU resources and has become a severe performance bottleneck in many backup storage systems [9].

Unfortunately, the single processor performance significantly impacts data deduplication [9]. To compensate for this performance drop, this paper indicates that data can be organized into classes in data deduplication systems according to the correlation strength between data files and using the k-mean algorithm. Furthermore, the processing of removing redundant data and the functional units are independent for each class. Therefore, it's a good idea to use parallel processing for each class, especially for chunking, calculating hash values, and comparing them [10]. The processors in modern computer systems are multicore to provide more Central Processes Unit (CPU) resources. Therefore, it can take full advantage of multicore computer systems with central processors to handle data deduplication that requires extensive computation [9]. Using parallelism in the CDC-based deduplication process increases the speed linearly, but the deduplication rate decreases slightly [9]. To address this problem, this paper proposes improving the Two Threshold Two Divisors (TTTD) algorithm by finding a list of divisors based on the data content and converting files belonging to each class into a stream of bytes. This leads to very few tail chunks and improves the deduplication ratio.

The rest of the paper is arranged into six sections, which are as follows: In section II, the relevant work is introduced. Then, In section III, the paper methods and tools are explained. Next, the dataset for examining the performance of the proposed method is presented in section IV, where the proposed method is presented in the V section, and the experimental results are discussed in the VI section. Finally, the conclusions are presented in the VII section.

II. RELATED WORK

Deduplication is the most extensively used technique for data backup and storage. As a result, deduplication systems have been the focus of several studies over the past few years. Here is a summary of some previous studies:

H. Fan et al. [11] 2019 proposed a new deduplication algorithm called Clustering Scattered Fingerprint CSF, where scattered fingerprints were collected. This method exploits the location of the data as much as possible and improves the rate of fingerprint usage by using clustering. The algorithm has also been improved by using the

scheduling strategy. The results showed that the proposed method works better than the modern algorithms.

H. Fan [9] 2019 presented a parallel method to speed up the deduplication process by dividing the deduplication process into four stages (chinking, fingerprinting, indexing, and writing). This proposed method significantly eased the bottleneck in processing chunking files and fingerprints and saving the data in storage. Experimental results indicate an acceleration of parallel CDC-based throughput. Also, the data deduplication rate decreased by 0.02%, a tiny percentage compared to the speed achieved by parallel CDC.

P. Sangat et al. [12] 2020 introduced a new effective approach to big data analysis by proposing a parallel algorithm known as ATrie Group Join (ATGJ). This algorithm integrates (join, grouping, and aggregation) operations to speed up the work of data analysis and reduce the burden on memory. According to the analysis of the algorithm, it scales better and is significantly quicker than other algorithms for handling concurrent threads, aggregate attributes, large data sets, and complex queries.

C. Ordonez et al. [13] 2020 proposed a low-cost processing method based on dynamically segmenting data sets during runtime. Each node processes one section of the data independently of the rest. After the processing is completed, the results of all the nodes are collected in the central processing node. Take advantage of parallel processing in the cloud, where a dynamic number of virtual processors is selected at runtime or when analyzing a data set for a short period, which can be applied to solve Machin Learning problems in big data analytics.

Ahmed Sardar M et al. [14] 2021 developed a new method for improving the redundancy deduplication system. They use a practical, boundary-based linear hash technique. Compared to Message-Digest Algorithm MD5 and Secure Hash Algorithm SHA-1, the new method reduces hash time by more than two times. Furthermore, the hash index table shrinks by 50% as a result of this.

R. Gururaj et al. [15] 2022 proposed using an inline data duplication system equipped with a cache eviction mechanism based on machine learning. It helps to reduce metadata overhead, eliminate repeated writes, and boost performance in storage applications. For example, exploiting block similarity eliminates 33% of unnecessary writes and reduces 54.5% of metadata overhead.

III. METHODS AND TOOLS

All the studies in the related work mentioned in this paper remove data redundancy using deduplication without paying attention to the type of files. That is, these studies treat the data regardless of the internal structure of the data. In the proposed method, the files are classified according to the internal structure of the bytes within these files. Then each class is dealt with separately regarding chunking, hashing, and comparison. This paper divided the data input into many classes and used parallel tasks to process each class independently. The proposed method improves the deduplication ratio and processing speed. Several algorithms were relied upon to complete this work. The most important of these algorithms are shown in this section.

1) ***Two Threshold Two Divisors (TTTD)*:** This algorithm works better than older algorithms by invalidating abnormally large chunks using the fallback divisor to reduce chunk sizes. This algorithm introduces a maximum and minimum threshold value to reduce the variance between the chunks [16]. When the chunk size reaches the maximum value threshold, the algorithm uses the fallback divisor and reruns the process to find a divisor representing a specific content boundary. However, this algorithm suffers from two main problems. The first is that these operations increase the number of fragments and the processing time of the deduplication process. The second problem is finding the second divisor, which wastes unnecessary time finding other backup breakpoints [17].

2) ***Statistical Features*:** This paper used three statistical measures to describe a particular file's features: Shannon's Entropy, Stander Division, and Variance. The experiments have shown that Entropy alone can be appropriate to determine the selected data set required in this study; However, we introduce Entropy, Stander Division, and variance to get better results. Entropy [18] is one of the most important techniques used to measure the amount of disorder and randomness in the information that makes up a particular file. It provides the amount of information and randomness in the given data regarding the number of bytes per file [19].

$$H(x) = -\sum_{i=1}^{n} p(x_i) log_2 p(x_i) \qquad (1)$$

X is a data vector, n is the alphabet size, and i is the number of bits. The function $p(x\ )$ is the probability of occurrence of byte value I in the segment [18].

3) ***Pearson's correlation coefficient (R)*:** is a metric for determining the strength and direction of the connection between two variables assessed at least on an interval scale. Pearson correlation coefficient is determined using (2):

$$R = \frac{N\sum xy - \bar{x}\bar{y}}{\sqrt{N\sum x^2 - \bar{x}^2}\sqrt{N\sum y^2 - \bar{y}^2}} \qquad (2)$$

Where R is the Correlation Coefficient of Pearson, N equals the number of pairs of values, ∑xy is equal to the sum of the products of x and y, $\bar{x}$ is equal to the mean of the x values, $\bar{y}$ is equal to the mean of the values y, $\bar{x}\bar{y}$ the Product of the mean values, ∑x2 the sum of the squares of all x values, ∑y2 the sum of the squares of all y values [20].

4) ***K-Means:*** Divide data into clusters that share similarities and differences from data from another

cluster. Using the K-Means clustering algorithm requires some steps, including the following [21]:

Step 1: Select the K value to determine the number of clusters.

Step 2: Choose a random K centroid.

Step 3: Place each data point inside one of the K clusters based on its proximity to the data's centroid.

Step 4: Create a new centroid for each cluster based on the variance calculation.

Step 5: Repeat step three and reassign all data points to the new nearest centroid point for each cluster.

Step 6: If there is a change, go to step 4; otherwise, the algorithm stops [22].

IV. DATASET

Several tests were conducted to examine the proposed method's performance based on three datasets from open-source systems .The datasets have different characteristics, and files belonging to datasets are of different sizes and types. The first dataset includes several (three-dimensional drawings plus the initials of the author Laurence D. Finston) 3DLDF files of (GNU's Not Unix) GNU source code versions comprising 5,795 files with a data size of 2.27 GB. The second dataset includes 309 versions of SQLite with a file count of 212,741 and a data size of 6.44 GB. Meanwhile, the third dataset, three versions of Linux source code Linux-4.9.311, Linux-4.19.239, and Linux-5.4.190, contains 183,779 Files and a data size of 2.17 GB [14].

ACKNOWLEDGMENT

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g.” Try to avoid the stilted expression, “One of us (R. B. G.) thanks …” Instead, try “R.B.G. thanks …” Put sponsor acknowledgments in the unnumbered footnote on the first page.

V. PROPOSED METHOD

The fundamental issue with the data deduplication technique is that it needs a costly computing effort to eliminate duplicate data due to the massive amount of big data. Therefore, this article aims to shorten the computation and time needed to complete the data deduplication stages. The chunking and hashing stages often need the most calculations and time. The proposed method removes redundant data by classifying the input data into several classes. It uses the parallel algorithm to remove redundant data based on task parallelism, the parallelism of independent data, and asynchrony. Furthermore, it uses parallel to speed up the elimination of duplicate data. Data deduplication processes are divided into four stages (chunking, fingerprinting, indexing, and writing). This method relieves the bottleneck of the deduplication system, especially in the chunking and fragmentation stages. Files belonging to each class are processed sequentially after being converted into a stream of bytes. Figure (1) describes the general framework of the proposed method.

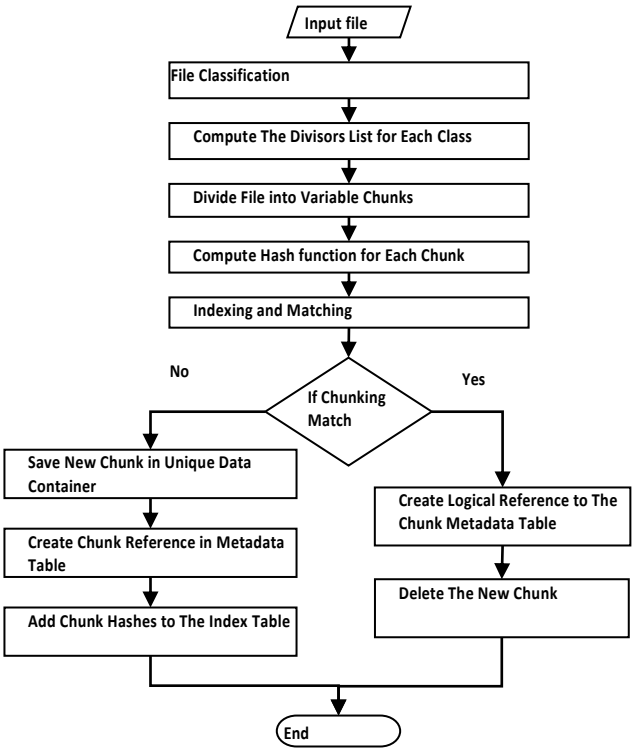


Fig. (1) The proposed method

A. Files classification

After the dataset is passed to the data deduplication system, the files are initially classified according to their file extension, resulting in several classes equal to the number of extensions. As a result, some classes are large or small. To achieve a reasonable number of classes, the highest-size extension classes are divided into lower-size subclasses, and the small extension classes are merged. The proposed algorithm tries to make the size of classes as close as possible based on the threshold value, as when the classes are close in size, this increases the performance in parallel processing. Figure (2) show classified data into several classes.

Divide Large Extension Class: Larger extension classes are divided into smaller subclasses. This method is used if the extension class size exceeds the threshold value. The extension classes are divided into several subclasses. Some statistical features are extracted from each file that belongs to subclasses. First, three features (Entropy, standard deviation, and variance) are computed for each file belonging to a given extension. Next, the K-mean algorithm clusters the files with similar features: files with Approximately the Same Entropy, variance, and stander division gather in the same cluster.

Merge Small Extension Class: This method aggregates extension classes of small size into classes of larger size. The sizes of the resulting classes are as close as possible. First, a histogram is calculated for each extension class. Next, the strength of the correlation coefficient between the histograms is calculated. Finally, using a Pearson matrix for grouped extension classes with a strong correlation into a single class.

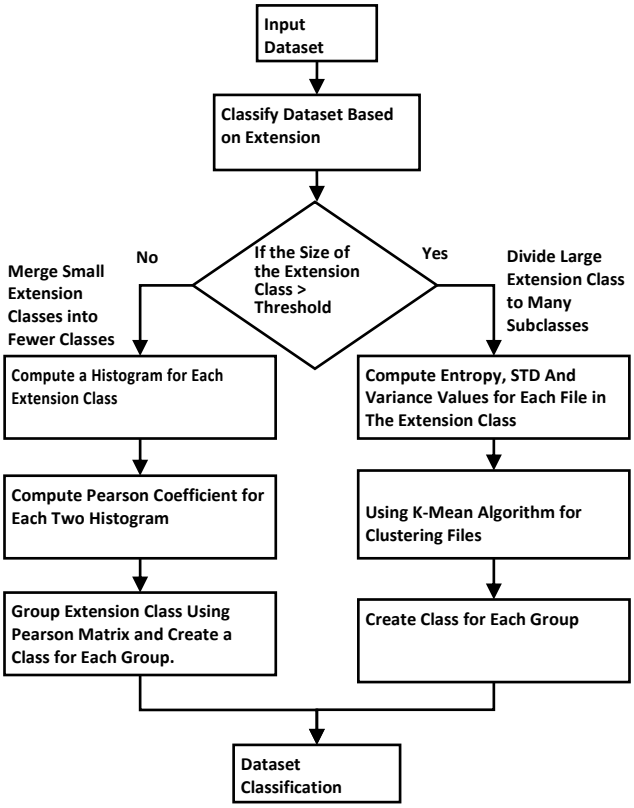


Fig. (2) Classifying data into several classes

B. Parallel Processing

In the data deduplication process, the chunking and the hashing are very costly in terms of time-consuming and process computations. The proposed method is designed to make efficient use of multicore processor computing. CDC-based deduplication systems generally have four stages (chunking, fingerprinting, indexing, and writing). These stages operate dependently on each other. Therefore, it can be processed sequentially in each data class.

The dataset is divided into classes, and each class is assigned one task for a parallel process. For example, all files belonging to Class 1 can perform all subtasks on them(data chunking, finding proper divisors, fingerprinting, and indexing) until they are stored on the hard drive. All class operations are performed entirely independently from the rest of the classes. First, all files belonging to the class are converted to a stream of bytes. Then, a sequence of bytes is generated for each class. The class size is usually more than 1 GB; if it's, buffer storage is used to deal with these classes. Figure (3) shows the parallel processing for each class independently.

1) LIST OF DIVISORS

The new method proposed a set of divisors for each class where a list of divisors is computed based on the frequency of the byte pairs within each class. This list of divisors is derived for each class based on the properties of the files that belong to that class. For example, some byte pairs have more frequency through statistical analysis than other pairs. Therefore, it starts by counting the number of times each pair of bytes appears or occurs in a given class.

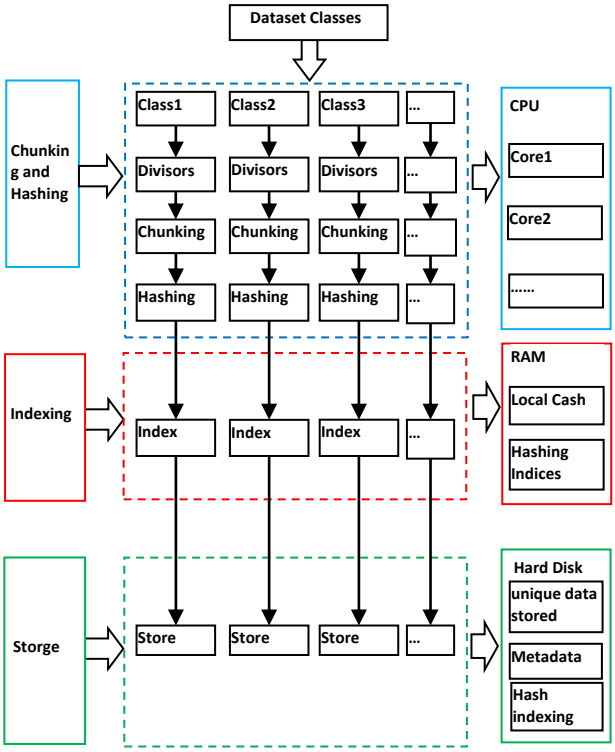


Fig. (3) Parallel processing for each class independently

Then the list of divisors is extracted according to the frequency of the byte pairs of each class, and this list is ordered from the highest frequency to the lowest frequency in descending order. Finally, each class defines the number of pairs of high-order bytes as divisors. The optimal number of divisors gives the best results determined by the experiment. Choosing the correct and perfect number of divisors required for each class improves deduplication results and increases DR.

2) CHUNKING

A new chunking method to improve The Two Divisors Two Threshold TTTD algorithm is proposed, where a list of D-divisors for each class in the dataset based on the dataset's contents is used to divide the stream of the input class into small, non-overlapping parts. It converts all files belonging to one class to a stream of bytes. A sequence of bytes is generated for each class. This leads to very few tail chunks, equal to the number of classes, only one tail chunk for each class. This method improves the deduplication ratio. Instead of using two divisors of (TTTD), the proposed method generates a list of divisors based on each class's content. These dividers define breakpoints to divide files into parts called fragments.

The new technique employs the minimum (Tmin) to decrease the number of too-small chunks and the maximum (Tmax) to decrease the number of too-large chunks. Deviser D is used to establish breakpoints. The scan begins with the minimum threshold value (Tmin) and ends with the highest threshold value (Tmax) to discover divisors by comparing each pair of bytes with the list of divisors and establishing breakpoints. If no breakpoints are given, the value (Tmax) is used as the breakpoint. Unlike prior approaches that divided files into fixed or variable sizes, the new method selects the divisors based on categorizing data sets, file formats, and the

frequency of byte pair occurrences, producing better deduplication results. Figure 4 shows that the chunk size between the last breakpoint and the class endpoint can be greater than 0 bytes and less than or equal to the maximum value (Tmax).

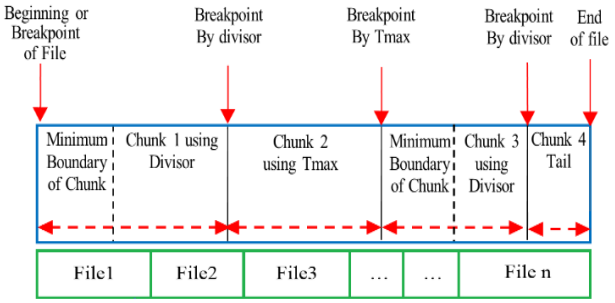


Fig. (4) Divide the stream of bytes in the class into Chunks based on Tmin,Tmax, and divisors

3) HASHING

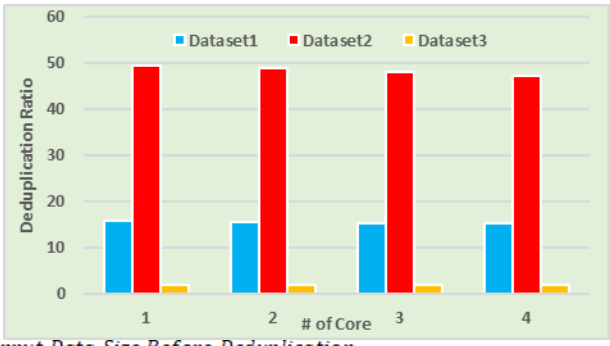
Identifying duplicate chunks of data quickly is one of the most critical challenges in deduplicating data. Finding duplicate data by comparing bytes is useless and requires many input and output operations. Most data deduplication systems rely mainly on fingerprinting data to remove duplicate data. Fingerprint functions are the most important procedure in data removal. Fingerprints are the same if and only if the data chunk is the same. The MD5 and SHA-1 Functions are among the known functions that are used for fingerprinting. The MD5 function provides faster computation than the SHA-1 function, which has a more negligible collision probability. The MD5 hash is 128 bits, and SHA-1 is 160 bits. In this paper, we use MD5 as the fingerprinting function. Parallel processing is used to speed up the fingerprinting procedure.

4) INDEXING AND WRITHING

The comparing process includes the divisor, size, and MD5 hash function of two chunks from the same class to find the identical. In previous deduplication techniques, the chunk is compared with all other chunks in the hash index table and is not considered a file type; therefore, these methods are inefficient and time-consuming. The new technique speed up the comparison procedure as a new chunk is compared with chunks of the same class. In addition, parallelism at this level expedites the comparing and storing processes. When two chunks match, the metadata that defines the current chunk is updated, and the new chunk is removed. If the new chunk is not duplicated, the new chunk information is added to the metadata table, and then the new chunk is placed in the non-duplicate data container.

5) EXPERIMENTAL AND RESULT

The proposed method is evaluated using the following computer characteristics: Windows 11 operating system, CPU Intel(R) Core i5-10300H with 4 cores, Random-Access Memory RAM 16 GB, and programming language C# Visual Studio 2022. In addition, three criteria were used to measure efficiency: the time required to finish the deduplication process, the data Deduplication Rate (DR), as mentioned in (3), and throughput, which reflects the quantity of data the system can process in a given period, as shown in (4).



$$DR = \frac{Total\ Input\ Data\ Size\ Before\ Deduplication}{Total\ Input\ Data\ Size\ After\ Deduplication} \quad (3)$$

$$throughput = \frac{Processed\ Data\ MB}{Time\ in\ Second} \quad (4)$$

Parallel processing has been widely used to improve the efficiency of software systems. For example, multicore processors can speed up the ability to remove redundant data using a parallel process. The proposed method uses task parallelism, the parallelism of independent data, and asynchrony. In the beginning, data is classified before entering into the deduplication system, and parallel processing saves storage space and reduces the time used by the deduplication ratio. Figure (5) shows the relationship between the throughput and the number of processor cores. The throughput reaches the maximum when using 4 parallel cores to process the four stages of data deduplication (chunking, fingerprinting, indexing and writing). According to the experiment's findings, most chunks were decided by the divisor when chunk sizes were between 128 and 512 bytes for the Tmin and Tmax values.

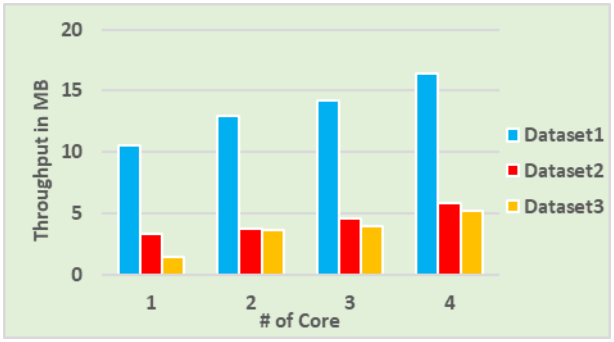


Fig. (5) Relationship of Throughput and Task Number

Figure (6) shows that each class is processed using one processor core. Increasing the number of classes reduces the time required to remove redundant data because each class is processed simultaneously with the rest. For example, each dataset is classified into 1,2,3,4 classes using histogram and k-mean algorithms. Therefore, when using 4 cores, the redundant removal data processing time is less than in other cases (1,2 and 3 cores).

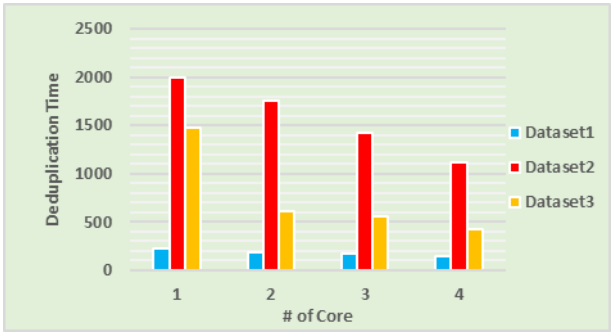


Fig. (6) Comparison of the Number Core and Deduplication Time

The deduplication ratio dropped slightly when the number of cores increased due to the data division into classes and the matching of similar chunks in each class separately. However, the increase in speed obtained from applying the proposed method is much more than a simple decrease in the deduplication ratio. Figure (7) shows the relationship between the deduplication ratio and the number of cores.

Fig. (7) Comparison of the Number Core and Deduplication Ratio

The proposed chunking algorithm proved its efficiency by the results in Table 1, where most chunks are obtained using the list of divisors generated using the proposed method. The divisor list is determined based on the dataset's properties. While the divisors obtained using (Tmax) were few. Also, all files of the same class are converted to a stream of bytes; this reduces the number of chunks of type (tail), while in the case of converting each file to a stream of data individually, the number of chunks of type (tail) is enormous. Furthermore, the number of chunks (tail) was one for each class, which increased the data deduplication ratio.
The performance of the proposed method has been compared with two published deduplication approaches. The results revealed the superiority of the proposed methods in distinguishing more duplicate chunks as efficiently as possible compared to the latest versions. Table II shows comparing the proposed method with the two methods,

TABLE I. NUMBER OF DIVISORS, TMIN, TMAX AND TAIL

Dataset	Type of Stream	Tmin	Tmax	divisors	tail
Dataset1	File	15	903,265	14,130,831	5,780
	Class	0	903,916	14,133,099	4
Dataset2	File	733	1,281,967	40,733,806	189,970
	Class	0	1,282,342	40,825,449	4
Dataset3	File	4,243	18,428	16,080,753	179,707
	Class	0	19,649	16,149,337	4

TTTD and BSW, for the Deduplication Ratio.

TABLE II. COMPARING THE PROPOSED METHOD WITH THE TWO METHOD

Dataset	*TTTD*	*BSW*	*Proposed Method*
Dataset1	2.13	1.7	15
Dataset2	16.7	13.5	35

Figure (8) shows the Deduplication Ratio for the proposed method compared to the other methods, TTTD and BSW.

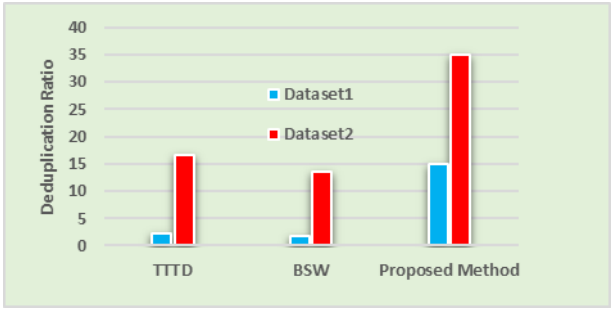


Fig. (8) Comparison proposed method with TTTD and BSW methods

6) CONCLUSION

The data deduplication technique's fundamental problem is time-consuming and requires expensive computing effort. This work presents a new method to speed up the deduplication process using parallel processing by taking advantage of multicore processors. It removes redundant data by classifying the input into several classes using the histogram similarity and k-mean algorithm. The proposed solution ensures high performance of deduplication operations and efficient use of system resources. Experimental evaluation showed that the proposed data deduplication method yielded more effective and faster results than the other approach, which does not rely on data classification and parallel processing. When using a processor with four cores, the speed is 55% to 250% faster than when using a single-core processor; also, throughput is 45% to 200% higher. Furthermore, generating the list of divisors for each class based on the content of files and converting each class to one stream of bytes increased the deduplication ratio. The proposed method is about 7 times higher than that of TTTD and about 10 times higher than BSW. In future work, the proposed method can be extended by dealing with real datasets and using a dynamic method for defining the divisor list for each class used in dividing the

files into chunks to increase the efficiency of eliminating redundant data.

REFERENCES

[1] G. Sujatha and J. R. Raj, “A Comprehensive Study of Different Types of Deduplication Technique in Various Dimensions,” *A Compr. Study Differ. Types Deduplication Tech. Var. Dimens.*, vol. 13, no. 3, pp. 316–324, 2022.

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

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

[4] N. Sharma, A. V. Krishna Prasad, and V. Kakulapati, “Data deduplication techniques for big data storage systems,” *Int. J. Innov. Technol. Explor. Eng.*, vol. 8, no. 10, pp. 1145–1150, 2019.

[5] P. Prajapati and P. Shah, “A Review on Secure Data Deduplication: Cloud Storage Security Issue,” *J. King Saud Univ. - Comput. Inf. Sci.*, no. xxxx, 2020.

[6] S. T. Ahmed and L. E. George, "Lightweight hash-based deduplication system using the self-detection of most repeated patterns as chunks divisors," *J. King Saud Univ. - Comput. Inf. Sci.*, no. xxxx, 2021, doi: 10.1016/j.jksuci.2021.04.005.

[7] F. Ni, X. Lin, and S. Jiang, “SS-CDC: A two-stage parallel content-defined chunking for deduplicating backup storage,” in *SYSTOR 2019 - Proceedings of the 12th ACM International Systems and Storage Conference*, 2019, pp. 86–96.

[8] F. Ni, X. Lin, and S. Jiang, “SS-CDC: A two-stage parallel content-defined chunking for deduplicating backup storage,” *SYSTOR 2019 - Proc. 12th ACM Int. Syst. Storage Conf.*, pp. 86–96, 2019, doi: 10.1145/3319647.3325834.

[9] W. Xia, D. Feng, H. Jiang, Y. Zhang, V. Chang, and X. Zou, “Accelerating content-defined-chunking based data deduplication by exploiting parallelism,” *Futur. Gener. Comput. Syst.*, vol. 98, pp. 406–418, 2019, doi: 10.1016/j.future.2019.02.008.

[10] P. Sobe, D. Pazak, and M. Stiehr, “Parallel Processing for Data Deduplication,” *PARS-Mitteilungen*, vol. 32, pp. 109–118, 1AD.

[11] H. Fan, G. Xu, Y. Zhang, L. Yuan, and Y. Xue, “CSF: An efficient parallel deduplication algorithm by clustering scattered fingerprints,” *Proc. - 2019 IEEE Intl Conf Parallel Distrib. Process. with Appl. Big Data Cloud Comput. Sustain. Comput. Commun. Soc. Comput. Networking, ISPA/BDCloud/SustainCom/SocialCom 2019*, pp. 602–607, 2019.

[12] P. Sangat, D. Taniar, and C. Messom, “ATrie Group Join: A Parallel Star Group Join and Aggregation for In-Memory Column-Stores,” *IEEE Trans. Big Data*, vol. 8, no. 4, pp. 1020–1033, 2020.

[13] C. Ordonez, S. T. Al-Amin, and X. Zhou, "A Simple Low-Cost Parallel Architecture for Big Data Analytics," *Proc. - 2020 IEEE Int. Conf. Big Data, Big Data 2020*, pp. 2827–2832, 2020.

[14] A. S. M. Saeed and L. E. George, “Fingerprint-based data deduplication using a mathematical bounded linear hash function,” *Symmetry (Basel).*, vol. 13, no. 11, pp. 1–19, 2021, doi: 10.3390/sym13111978.

[15] R. Gururaj, M. Moh, T. S. Moh, P. Shilane, and B. Bhanjois, “Performance Centric Primary Storage Deduplication Systems Exploiting Caching and Block Similarity,” 2022.

[16] K. Eshghi and H. K. Tang, “A framework for analyzing and improving content-based chunking algorithms,” *Hewlett-Packard Labs Tech. Rep. TR*, no. August, 2005, [Online]. Available: http://shiftleft.com/mirrors/www.hpl.hp.com/techreports/2005/HPL-2005-30R1.pdf%5Cnpapers3://publication/uuid/053B1556-804C-4F39-BD0B-2EBD9C047F30

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

[18] J. Natal, I. Ávila, V. B. Tsukahara, M. Pinheiro, and C. D. Maciel, “Entropy: From thermodynamics to information processing,” *Entropy*, vol. 23, no. 10, pp. 1–14, 2021, doi: 10.3390/e23101340.

[19] Y. Zhang *et al.*, “A Fast Asymmetric Extremum Content Defined Chunking Algorithm for Data Deduplication in Backup Storage Systems,” *IEEE Trans. Comput.*, vol. 66, no. 2, pp. 199–211, 2017.

[20] E. Isaac and E. Chikweru, “Test for Significance of Pearson’s Correlation Coefficient,” *Int. J. Innov. Math. Stat. Energy Policies*, vol. 6, no. 1, pp. 11–23, 2018, [Online]. Available: www.seahipaj.org

[21] X. Chu, J. Lei, X. Liu, and Z. Wang, “KMEANS Algorithm Clustering for Massive AIS Data Based on the Spark Platform,” *2020 5th Int. Conf. Control. Robot. Cybern. CRC 2020*, pp. 36–39, 2020, doi: 10.1109/CRC51253.2020.9253451.

[22] Z. Wang, Y. Zhou, and G. Li, “Anomaly Detection by Using Streaming K-Means and Batch K-Means,” *2020 5th IEEE Int. Conf. Big Data Anal. ICBDA 2020*, pp. 11–17, 2020.