New Techniques to Enhance Data Deduplication using Content based-TTTD Chunking Algorithm

Hala AbdulSalam Jasim, Assmaa A. Fahad
Department of Computer Science, College of Science
University of Baghdad
Baghdad, Iraq

***Abstract*—Due to the fast indiscriminate increase of digital data, data reduction has acquired increasing concentration and became a popular approach in large-scale storage systems. One of the most effective approaches for data reduction is Data Deduplication technique in which the redundant data at the file or sub-file level is detected and identifies by using a hash algorithm. Data Deduplication showed that it was much more efficient than the conventional compression technique in large-scale storage systems in terms of space reduction. Two Threshold Two Divisor (TTTD) chunking algorithm is one of the popular chunking algorithm used in deduplication. This algorithm needs time and many system resources to compute its chunk boundary. This paper presents new techniques to enhance TTTD chunking algorithm using a new fingerprint function, a multi-level hashing and matching technique, new indexing technique to store the Metadata. These new techniques consist of four hashing algorithm to solve the collision problem and adding a new chunk condition to the TTTD chunking conditions in order to increase the number of the small chunks which leads to increasing the Deduplication Ratio. This enhancement improves the Deduplication Ratio produced by TTTD algorithm and reduces the system resources needed by this algorithm. The proposed algorithm is tested in terms of Deduplication Ratio, execution time, and Metadata size.**

***Keywords*—Data deduplication; big data compression; data reduction; Two Threshold Two Divisor (TTTD); chunking algorithm**

I. Introduction

There is an explosion on the amount of digital data in the world right now, as manifest by the considerable growth in the measured amount of stored data in 2010 and 2011 from 1.2 zettabytes to 1.8 zettabytes [1], respectively [1], and the prophesied amount of data to be created in 2020 is 44 zettabytes [2], [3]. So manage the storage which is cost-effectively, has become an important task of the most challenging in the big data era. The workload studies performed by an American multinational corporation Dell, EMC, (Richard Egan, Roger Marino & John Curly the E, M & C in EMC) and Microsoft, suggest that approximately 50% and 85% of the data are redundant in their primary and secondary storage systems, respectively.

According to International Data Corporation (IDC) recent study, almost 80% of the surveyed corporations indicated that they are using in their storage systems to reduce redundant data kind of data deduplication technologies, which increased storage in an efficient way and reduced the costs of storage spaces [4].

Data deduplication does not only reduced storage space, but also decreased the transmission rate by eliminating redundant data in low bandwidth network environments. A sub file-level chunking deduplication system breaks the input data stream into multiple data “chunks” that are individually distinguished by a hash signature (e.g., SHA-1), and detects the duplicate ones by some kind of comparison method. Deduplication systems remove duplicate chunks, and store or transfer only one copy of them to achieve the goal of saving storage space or network bandwidth. In the other hand Deduplication system, suffers from the long execution time and the need of many CPU resources on its job.

Teng-Sheng Moh [5] in 2010 adds a new switch condition to enhance the execution time of TTTD algorithm with the same deduplication ratio. He reduced the value of the main divisor (D) and the second divisor (Ddash) to the half when the break point was not found before 1600 byte, this condition reduced about 6% of the running time and 50% of the large-sized chunks.

Manogar and Abirami [6] in 2014 first examined and compared different deduplication techniques, and then they concluded that variable size data deduplication is more efficient than the rest of the deduplication techniques.

AbdulSalam and Fahad [7], in 2017 performed a survey on different chunking algorithms of data deduplication. They discussed, studied the most popular chunking algorithm TTTD, and evaluated this algorithm using three different hashing functions; Rabin Finger print, Adler, and SHA1 implemented each one as a fingerprinting and hashing algorithm and then compared the execution time and deduplication elimination ratio.

In this paper a new chunking condition is added to enhance the deduplication ratio and a new four hashing function is proposed to improve the matching process by reducing the probability of hash collision occurrence. In addition, a searching technique suggested in order to reducing the time needed for deduplication process.

[1] IDC, “The 2011 digital universe study,” Tech. Rep., Jun. 2010, [Online]. Available: http://www.emc.com/collateral/analystreports/idc-extracting-value-fromchaos-ar.pdf

TABLE I. CHARACTERISTICS OF THE USED DATA SET

Data Set	On Line Link	Number of Files and Folders	Total Size
Versions of Emacs of GNU	http://www.gnu.org	16,296 Files, 327 Folders	580 MB
Versions of 3DLDF of GNU	https://www.kernel.org	5,795 Files, 63 Folders	1.27 GB

The input data to the system consists of a number of files with diverse sizes and types. The system process theses files as one file at a time. The proposed system is developed and tested on two data sets, which belongs to the GNU file system in order to show the efficiency of the proposed system. Table I shows the characteristic of each data set.

II. DATA DEDUPLICATION SYSTEM USING TTTD CHUNKING ALGORITHM

In general, any deduplication system will pass into three stages: (Chunking, Hashing and Indexing, and Matching stage). The theory of each part will explain briefly:

A. Chunking

The first step of data deduplication is chunking, it partitioning the input data stream (file) into small and non-overlapping parts named chunks. The chunking operation is performed using certain type of rolling hash that depends on the contents of the text itself so that for two strings with the same contents it will produce the same hash value. This stage is a very important stage; the deduplication ratio depends on the chunks produced from this stage [7].

TTTD is a variable size-chunking algorithm [8], it use Rabin Fingerprint to find the hash value of substring with predefined window size (48 byte). If the hash of this substring satisfy the condition of TTTD it will considered as a breakpoint otherwise slide the window size one byte [9].

Formula (1) used to compute Rabin Fingerprint for the first substring. Then, Formula (2) used for the rest substring, worked by remove first character, and added the new one [4].

$$\mathrm{Rabin}(B_1, B_2, \ldots B_\alpha) = \{\sum_{i=0}^{\alpha} (B_i * P^{\alpha-i-1})\} \text{ Mod } D \tag{1}$$

$$\left\{\left[\mathrm{Rabin}(B_i, B_{i+1}, \ldots B_{i+\alpha-1}) - B_i * P^{\alpha-1}\right] * P + B_{i+\alpha}\right\} ModD \tag{2}$$

Here: D is the average chunk size [7], Bx is the ASCII code for the substring characters, P is a prime number α is the size of the sliding window.

B. Hashing and Indexing

The main target of hashing and indexing stage is to compute the hash value for whole chunk and adding it to the lookup table or index table. When the finger print satisfy one of TTTD conditions then the whole chunk of text will be sent to the Hash function (like SHA-1 or MD5).The hash value result from the hash function will be used to compare between the chunks. The name of the chunk is the location that saved with inside the chunk container. The content of the lookup table will be a set of records consist of two fields, the first field contain the chunk name, and the second field will be contain its hash value:

D:\DataBase_test\emacs-22.1\AUTHORS\Chunk-0.txt
123456

C. Matching

In original systems, the chunk of new file will compared to the chunk of the files that have the same name and type. If there is a matching then the system will retrieve its lookup table, and compare all the chunks of the new file with the chunks of the old one. For the duplicated chunks, delete the new chunk and perform a logical reference to the old one in a final lookup table of the new file, otherwise save the chunk, and add its name and its hash to the final lookup table. A collision problem may occur during the matching operations, to solve the collision problem a byte-to-byte comparison must be performed [10]. The number of collision reduces the performance of the system due to the time needed to solve it. Reducing the number of collision is one of the aims of a good deduplication system.

III. PROPOSED SYSTEM AND METHOD

In this paper, a Content Based Two Threshold Two Divisor with Multi-Level Hashing Technique (CB-TTTD-Multi-Level Hashing Technique) based on TTTD algorithm suggested to enhance deduplication technique by speed up the deduplication operation and increase its compression ratio.

A. Chunking

CB-TTTD-Multi-Level Hashing Technique introduces a new hash functions to compute the fingerprints for each tested data stream. For each character in the first string of window size (36 byte) the fingerprint value is calculated using (3). Then for each of the following substrings, fingerprint value is calculated using (4).

$$\mathrm{FingerPrint}\ (B_0, B_2, \ldots B_\alpha) = \{\sum_{i=0}^{\alpha-1} \mathrm{Val}[B_i] * 2^{i+1}\} \tag{3}$$

$$\text{New FingerPrint}\ (B_{i+1}, \ldots B_{i+\alpha+1}) = [\{\mathrm{FingerPrint}(B_i, \ldots B_{i+\alpha}) - \mathrm{Val}[B_i]\} \div 2\ ] + \mathrm{Val}[B_{i+\alpha+1}] * 2^{\alpha-1} \tag{4}$$

Here: α is Substring Size, B1 … Bα: are the substring characters, Val [Bi]: is the value of index [Bi] in Fingerprint-array. The value of character taken from an array of 256 position that represent the printable characters filled with random value of (1, 2) to produce an array as in Fig. 1:

Unlike Rabin fingerprint CB-TTTD-Multi-Level Hashing Technique, uses a value retrieved from the fingerprint array instead of using the ASCII code of the character, this step speed up the computation and reduce the overhead of CPU needed for each fingerprint because it uses very small values. The system test different values for the fingerprint array such as:

[0,1] , [1,0] , [0 , 0, 1,1] , [1,1,0,0] , [1,2] , [1,2,3,1,2,3] , [1,2,3,4,…. 255].

Array [0]	Array [1]					Array [255]
1	2	1	2	1	………	2

Fig. 1. Fingerprint array.

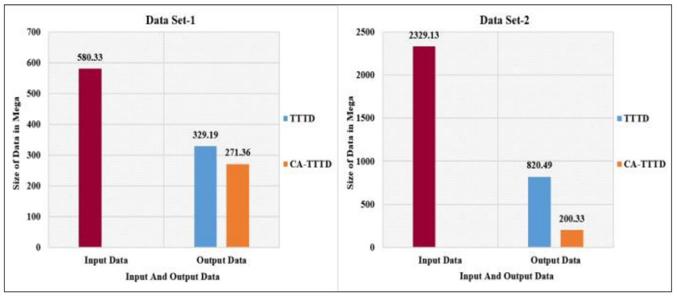

Fig. 2. Input to output data ratio of the two-compared algorithms.

The most efficient values that produces a high deduplication ratio was the sequence of [1, 2] values. This technique helps to produce different hash values for different substrings that helps in detection more redundant data and increase the deduplication ratio.

In addition, CB-TTTD-Multi-Level Hashing Technique gives a weight to the characters in the data stream. The system considers the Dot character ('.'), as a new condition, in addition to the main and second divisor condition of TTTD. When a '.' character is found followed by a (space) or (end of line) this paragraph is considered as a separated chunk. The advantage of this condition appears in case of two paragraphs considered as one chunk, then any change in one paragraph may affect the next one, but in this case, the effect will be limited with the changed paragraph only. Adding this condition increased the deduplication ratio and with the same chunking time, because it does not need any extra processing steps to compute the chunk boundary. Fig. 2 illustrates the output size using original TTTD chunking algorithm and CB-TTTD-Multi-Level Hashing Technique on the same dataset as input and the differences between them. Table II shows the result produced by implementing CB-TTTD-Multi-Level Hashing Technique on Dataset1 compared with TTTD algorithm.

B. *Hashing and Indexing*

The old deduplication system is suffering from the wasted time needed to solve the collision problem. CB-TTTD-Multi-Level Hashing Technique suggests a new method that uses four hashing functions rather than one to solve the collision problem. The technique will compute and save four hash values for each chunk, as shown in Table III Using the hash functions shown below:

$$Hash1(chunk) = \left\{ \sum_{i=0}^{s-1} (Array1[Bi] * 2^{k}) \right\} \quad (5)$$

$$Hash2(chunk) = \left\{ \sum_{i=0}^{s-1} (String[Bi] * A1) \,\&\, 0xFFFFFFFF \right\} \quad (6)$$

$$Hash3(chunk) = \left\{ \sum_{i=0}^{s-1} (Array3[Bi] * 2^{k}) \right\} \quad (7)$$

$$Hash4(chunk) = \left\{ \sum_{i=0}^{s-1} (String[Bi] * A2) \,\&\, 0xFFFFFFFF \right\} \quad (8)$$

TABLE II. THE EFFECT OF DOT CONDITION ON EACH SYSTEM

Algorithm	Number of Chunks	Deduplication Ratio	Size of Metadata in MB	Time in second
TTTD without Dot	621861	1.78300	82.1	2304
TTTD with Dot	1542374	1.81176	124	3349
Proposed System without Dot Condition	644933	2.03845	16.7	458
Proposed System with Dot Condition	960091	2.1386	24.1	582

TABLE III. ARRAY1 AND ARRAY2 VALUES

Array	[0]	[1]	[2]	[3]	[4]	…	[255]
Array1	0	1	2	3	4	…	256
Array2	0	1	0	1	0	…	1

Here: S is the chunk size, Array1 and Array2 is an array of 255 value as shown in Table III, K is an integer value equals to (i mod 8), A1 and A2 values is 5, 11 respectively, their values increased by one every iterator, 0xFFFFFFFF used to get limited range of value.

Using these hash functions reduces matching time by solving the collision problem in an efficient way, Also the number of bits needed to store these four hashes are about to 32 bits maximum, which is less than the number of bites needed to save the hash value in SHA-1and MD5 which yields hexadecimal digits, SHA-1 returning 160 bit. 4 bit per character and thus equals to 40 character, and the output of MD5 hash which is 128 bits equals to 32 characters [11].

The name, the size, and the four hashes values for each chunk must save in Index-Table. The name of the chunks in CB-TTTD-Multi-Level Hashing Technique is an integer number from zero to N, where N is unlimited number increased with each chunk in the system. This information must be ordered in the Index table as shown in Table IV.

CB-TTTD-Multi-Level Hashing Technique also creates a log file for each file in the dataset. This log file will be used in reconstruction operations of the files.

TABLE IV. THE STRUCTURE OF THE INDEX-TABLE

Chunk Name	Chunk Size	Hash1	Hash2	Hash 3	Hash4
0	1268	357315 1	7173770 3	19983	7241537 3
1	466	129976 9	9843858	6961	1009266 6
2	480	133838 6	1050308 0	7454	1076210 0

TABLE V. TIME AND INDEX TABLE IMPACT OF EACH HASH

Number of Hash Used	1-Hash	2-Hashes	3-Hashes	4 -Hashes
Chunking Time in Second	353	342	321	336
Deduplication Time in Sec	1050	656	627	562
Size of Log File in MB	6.048	6.048	6.048	6.048
Size of Index Table in MB	10.915	13.802	15.177	18.075
Final Meta Data Size in MB	16.963	19.85	21.225	24.123
Number of Hash Collision	21372	0	0	0

C. Matching

In deduplication matching steps, when a new file comes and passes the two previews stages, the system must detect and eliminate the duplicated chunks. It first check the hash values of the chunks, if the hash values are similar, then the algorithm will compare the two chunks byte to byte, if they are identical the system will delete the new one and add a logical reference to the location of the old one. Otherwise a collision was occurs; the chunks are difference; the system will save the new one as a new chunk. This operation takes a lot of time and overhead the system. Therefore, a new method has to add to deduplication matching process, to enhance the throughput, i.e., saving execution time and reduce CPU resources usage.

In this paper, the Multi-Level Hashing Technique was suggest to enhancing the matching process by using the four hash functions that already computed in hashing stage. If a collision occur in first hash values of the compared chunks then compare the second, third and fourth hash. The test result shows a significantly noteworthy improvement with the time needed by matching process, because comparing four numbers is faster than comparing the whole compared chunks byte-to-byte.

This solution tested with dataset1 and dataset2, the collision founds in first hash function will reduced to zero by the second hash function. For dataset1 with the first hash function the collision number was 21372 for total chunk number 960091, by using the second hash function, the number is reduce to zero. Third and fourth hash functions are uses as an extra step to be assurance the collision is determined, if a chunk overpasses the first two hashes. For each hash in the system the elapsed time and collision number is computed as shown in Table V, the proposed system found that four hashes is a balanced number between time and the size of index table.

When a new file comes, it must be chunked and hashed, and then each chunk will be compared with all chunks within the database. The previews deduplication systems search for the similarity of the chunks within a file name or type in the dataset, the proposed system search for the similarity of the chunk within the whole files in the dataset. The side effect of this method is the time needed to complete the matching operation. To enhance this method, the size of the chunk is utilized as the searching parameter, and a binary search technique is used instead of linear searching method. Because when using liner search, the system will be with O (N) complexity while using the binary search reduce the complexity to O (log N) [12].

To implement a binary search in an efficient way, the new matching algorithm that proposed in this paper, divide the chunks in the workspace into 16 groups, depending on the chunk size as shown in Table VI. The first group contains the chunks with size (0 – 462) byte, and the second group is for the chunks with size (463 – 471) byte, and so on. The number of the chunks in each group is approximately equal.

Fig. 3 shows chunks distribution, the minimum chunk size of the algorithm is 460 byte and the maximum chunk size is 2800 byte as the TTTD algorithm suggested [9]. However, there are special cases where the chunk size is less than 460 byte. These cases are:

- The size of the file is smaller than the minimum chunk size (460 byte) or less than window size (36 byte).
- The size of the rest of the file from the last breakpoint is less than 460 byte or 36 byte.
- The breakpoint could not found after the last breakpoint to the end of the file.

In this paper the data in the above three cases will be preserved as one chunk.

Working with large amount of the chunk as groups is easier and faster than working with it as a one large search space Table VII shows the effect of the searching techniques with respect to time.

TABLE VI. CHUNK DISTRIBUTION ACCORDING TO PARTITIONING METHOD

Index	From	To
0	0	462
1	463	471
2	472	481
3	482	491
4	491	503
5	504	515
6	516	530
7	531	547
8	548	570
9	571	598
10	599	636
11	637	691
12	692	775
13	776	920
14	921	1291
15	1292	2800

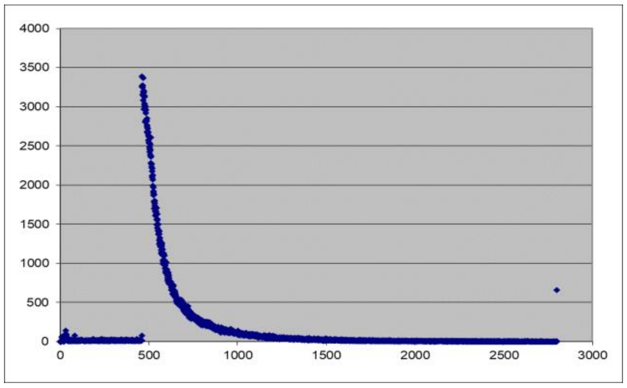

Fig. 3. Distribution of chunk with respect to size.

TABLE VII. COMPARISON BETWEEN SEARCHING WITH ONE BIG SEARCH SPACE AND PARTITIONS METHOD

Benchmarks	**One Large Search Space**	**16 - Parts Search Space**
Number of Chunk	960091	960091
Deduplication Ratio	2.1386	2.1386
Chunking Time in Second	320	339
Whole Program Time (Matching and Chunking) in Second	3230	629

To represent the chunk of dataset with a distribution-based representation that summarizes scalar information into much-reduced groups, one of statistical distribution methods should be used [13]. In this paper, CB-TTTD with Multi hashing Technique used the histogram. The disadvantages of used statistical distributions method is that the distribution representing the chunks in one dataset differs from another one.

However, in the proposed case the parts boundaries was approximately equals for all tested datasets. The histogram steps that used to rearrange the chunks in to range from (0 to16) parts depending on chunks size instead of (0-2800) range are:

- Count number of chunk for each size of range (0 – 2800).
- Compute the probability of each size with respect to other; i.e.: P (i) = (count (i)/∑ count (i)).
- Compute the Probability Ratio for each size using the formal: PA (i) = (∑ P (i)).
- Multiply the PA column with Density Slicing number which in our case is (15) and round the result to nearest integer number, that give as range from (0-15) only, see Table VI.

IV. RESULTS AND DISCUSSION

Proposed technique is implemented on a machine with configuration Intel i7 CPU with Installed memory 4.00 GB on 64bit windows OS. To implement proposed technique dataset is collected as mentioned in Table I. TTTD chunking algorithm with Rabin fingerprint and SHA-1 hashing algorithm implemented also in the same environment to compare the result of the proposed system with it.

To analyze CB-TTTD-Multi-Level Hashing Technique the following performance metrics are used. Table VII shows the result of the two algorithms.

- Data Size after Deduplication: It describes how many data remains after the data deduplication eliminates all redundant data.
- Deduplication Gain: It indicates how much unique content is present in the dataset. In this paper, it calculated as in (9).

$$\text{Deduplication Gain} = \frac{\text{The Size of Deduplicated Data Detected}}{\text{Total Output Data Size After Deduplication}} \qquad (9)$$

TABLE VIII. COMPARISON TTTD AND CB-TTTD-MULTI-LEVEL HASHING

Evaluation Metrics	**DataSet-1**		**DataSet-2**	
	TTTD	***CB-TTTD***	***TTTD***	***CB-TTTD***
Input Data (MB)	580.33	580.33	2329.13	2329.13
Data Size after Deduplication (MB)	329.19	271.36	820.49	200.33
Duplicated Data Detected (MB)	251.14	308.97	1508.64	2128.8
Deduplication Rate	1.7628	2.1386	2.8387	11.6264
Deduplication Gain	0.4327	0.5324	0.6477	0.9139
Chunking Time in Sec	727	462	3747	1075
Deduplication Time in Sec	2632	532	9083	1539
Total Number of chunk	621861	960091	2468026	2611322
Average Chunk size	978.55	633.82	989.56	935.26

- Deduplication Ratio: The data deduplication ratio measures the effectiveness of the deduplication process, it is expressed as in (10).

$$\text{Deduplication Ratio} = \frac{\text{Total Input Data Size Before Deduplication}}{\text{Total Input Data Size After Deduplication}} \qquad (10)$$

- Average Chunk size: Calculated as in (11)

$$\text{Average Chunk Size} = \frac{\text{Total Input Data Size}}{\text{Total Number of Chunks}} \qquad (11)$$

- Chunking and Hashing time: It is the total time taken to perform hashing and chunking operation.

Experimental results are shown in Table VIII. These results clearly demonstrate that CB-TTTD-Multi-Level Hashing Technique satisfactorily reduces the deduplication processing time and increases its ratio.

Result charts also clearly demonstrate that our proposed approach perform better than original approach for small and big data sets, the deduplication gain is also increased.

The proposed finger print equation used in chunking stage is more efficient than Rabin fingerprint equation; it increase the number of the chunks by the way it works, especially small chunk sizes, with less CPU overhead cost which increase the deduplication ratio. In addition, using Content Based condition (Dot character), also increased the number of the small chunks leading to increasing deduplication ratio without influence chunking time Fig. 4 illustrate the average chunks size of the two algorithms.

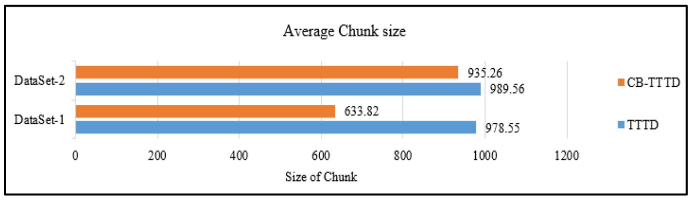

Fig. 4. Average chunk size.

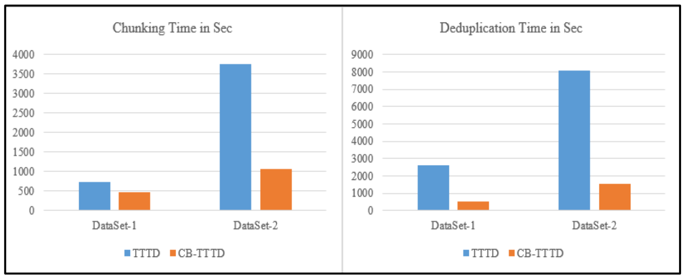

Fig. 5. Deduplication and chunking time.

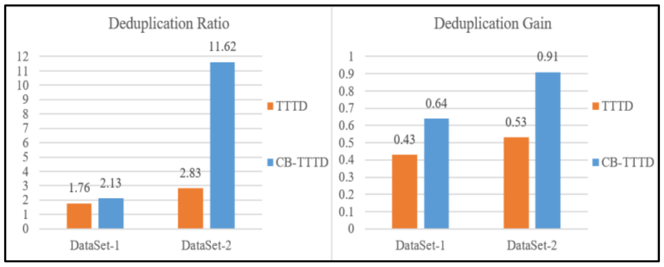

Fig. 6. Deduplication ratio and gain for both system.

The suggested comparing method improved deduplication ratio, but the execution time was increased. Nevertheless, this disadvantage of comparison was solved by partitioning the search space, and the use of mathematical modules to overcome the collision problem, which is an efficient solution that enhanced the execution time of matching process. As shown in Fig. 5, the time of chunking and the overall time (deduplication time) is less than the time of TTTD algorithm.

Fig. 6 depicts data deduplication gain and ratio. In proposed CB-TTTD-Multi-Level Hashing technique, data deduplication ratio and its gain is high as compared to traditional deduplication methods.

V. Conclusions and Future Work

In big data storage, data is too large and efficiently store data is difficult task. To efficiently stores and de-duplicate the data, this paper suggest a new technique to reduce the deduplication ratio. This technique examined the deduplication detection and elimination system performance and explained the rationale parts, data deduplication consist of three stages, the enhancement operation involved all stages that leads to good deduplication ratio and fast execution time, The Metadata produced by CB-TTTD with Multi hashing technique is less than the one that produced by traditional algorithms. The effectiveness of the proposed method was evaluated using two relative datasets; the preliminary results are encouraging to go forward toward developing new method for detection and elimination deduplication algorithms to meet the challenges and demands of fast and efficient deduplication systems. Moreover, we can use some kind of fast compression with the Meta data (Index Table and Log file) to saving more disk space.

References

[1] D. Stevenson and N. J. Wagoner, “Bargaining in the shadow of big data,” Fla. Law Rev., vol. 66, no. 5, p. 66, 2014.

[2] Gantz, John, and David Reinsel. "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, pp 1-16 , 2012.

[3] Turner, V., Gantz, J. F., Reinsel, D., & Minton, S. “The digital universe of opportunities: Rich data and the increasing value of the internet of things”. IDC Analyze the Future,p. 5, 2014.

[4] Xia, Wen and Jiang, Hong and Feng, Dan and Douglis, Fred and Shilane, Philip and Hua, Yu and Fu, Min and Zhang, Yucheng and Zhou, Yukun, “A comprehensive study of the past, present, and future of data deduplication,” Proc. IEEE, vol. 104, no. 9, pp. 1681–1710, 2016.

[5] T.-S. Moh and B. Chang, “A running time improvement for the two thresholds two divisors algorithm,” Proceedings of the 48th Annual Southeast Regional Conference on - ACM SE ’10. p. 1, 2010.

[6] E. Manogar and S. Abirami, “A study on data deduplication techniques for optimized storage,” 6th Int. Conf. Adv. Comput. ICoAC 2014, pp. 161–166, 2015.

[7] H. Abdulsalam and A. Fahad, "Evaluation of Two Thresholds Two Divisor chunking algorithm using Rabin fingerprint, Adler, and SHA-1 hashing algorithms," The Iraqi Journal of Science,. paper 4C.58, 2017.

[8] Nisha, T. R., S. Abirami, and E. Manohar. "Experimental study on chunking algorithms of data deduplication system on large scale data." In Proceedings of the International Conference on Soft Computing Systems, pp. 91-98. Springer India, 2016.

[9] K. Eshghi and H. K. Tang, “A framework for analyzing and improving content-based chunking algorithms,” Hewlett-Packard Labs Tech. Rep. TR, 2005.

[10] Chen, Zhengguo and Chen, Zhiguang and Xiao, Nong and Liu, Fang, “Nf-dedupe: a novel no-fingerprint deduplication scheme for flash-based ssds,” in 2015 IEEE Symposium on Computers and Communication (ISCC), pp. 588–594.

[11] Zhang, Yang and Wu, Yongwei and Yang, Guangwen, "Droplet: A distributed solution of data deduplication," in Proc - IEEE/ACM Int Work Grid Comput. 2012;114–21.

[12] D. S. KUSHWAHA and A. K. MISRA, “Data structures a programming approach with C,” 2nd ed. PHI Learning Pvt. Ltd., 2014.

[13] Wang, Ko-Chih and Lu, Kewei and Wei, Tzu-Hsuan and Shareef, Naeemand Shen, Han-Wei, “Statistical visualization and analysis of large data using a value-based spatial distribution,” in 2017 IEEE Pacific Visualization Symposium (PacificVis), 2017, pp. 161–170.