Webology, Volume 18, Special Issue on Information Retrieval and Web Search
April, 2021

Comparative Analysis for Content Defined Chunking Algorithms in Data Deduplication

D. Viji
Research Scholar, Department of Computer Science and Engineering, Sathyabama Institute of Science and Technology, Chennai, India.
E-mail: dviji2k@gmail.com

Dr.S. Revathy
Associate Professor, Department of Information Technology, Sathyabama Institute of Science and Technology, Chennai, India.
E-mail: ramesh.revathy@gmail.com

Received November 08, 2020; Accepted December 20, 2020
ISSN: 1735-188X
DOI: 10.14704/WEB/V18SI02/WEB18070

Abstract

Data deduplication works on eliminating redundant data and reducing storage consumption. Nowadays more data generated and it was stored in the cloud repeatedly, due to this large volume of storage will be consumed. Data deduplication tries to reduce data volumes disk space and network bandwidth can be to reduce costs and energy consumption for running storage systems. In the data deduplication method, data broken into small size of chunk or block. Hash ID will be calculated for all the blocks then it's compared with existing blocks for duplication. Blocks may be fixed or variable size, compared with a fixed size of block variable size chunking gives a better result. So the chunking process is the initial task of deduplication to get an optimal result. In this paper, we discussed various content defined chunking algorithms and their performance based on chunking properties like chunking speed, processing time, and throughput.

Keywords

Data Deduplication, Content-defined Chunking, Cloud Storage System.

Introduction

Now a day's amount data rapidly increasing in the cloud environment due to the usage of the internet, Smartphone's and social networking platforms like Facebook, Instagram, and Twitter, etc. as per the report of International Data Corporation (IDC) the data volume will be reached 35ZB by Kaur, R, 2020. Huge data on the storage system is very difficult to handle so that they try to remove the redundant data from the cloud environment to improve storage efficiency. Deduplication is the most important technique for removing unnecessary duplicates data and ensuring unique contents are stored in the storage systems. Deduplication system methods are varying from one storage environment to another proposed by Paulo, J, 2014. Primary system deduplication achieved 68% efficiency but in the secondary storage system 83%, so the backup deduplication system dramatically increased their storage efficiency proposed by Viji, D., & Revathy, S, 2019.

Data deduplication techniques is an elegant data compression technique to avoid replicated data stored in the storage environment. This technique identifies the duplicate data and stores only one unique data, duplicated data are a pointed through the pointer of unique data. The main focus of deduplication techniques to increase storage efficiency in the cloud environment. Different types of data are involved in storage systems like text data, and multimedia data. Every data has individual storage formats and different properties. Based on the type of data, deduplication methods also differ to identify and removing replicated content. So the type of data is the major factor for developing deduplication methods.

Based on the content of information deduplication methods divided into major two types data deduplication and multimedia deduplication. Data deduplication contains the text information, multimedia contains image and video. Fig. 1 represents the data deduplication process replicated blocks are removed and stored unique blocks by Sanyal, S. (2018). Data deduplication can be done in two ways file-level and block-level deduplication. File-level deduplication of the whole content of the file compared with another file using the hash value by Li, J., et al., (2018). It is a very simple method but this way of deduplication method not providing high efficiency. Block-level deduplication achieves better efficiency, in this entire file broken into blocks or chunks. Blocking is crucial to speed-up the deduplication of large datasets by Lavanya, R. et al., (2017). This procedure is also named as chunking by Nguyen, Q.N, 2018. Fig 1 shows the entire file divided into four different blocks B1, B2, B3, and B4.

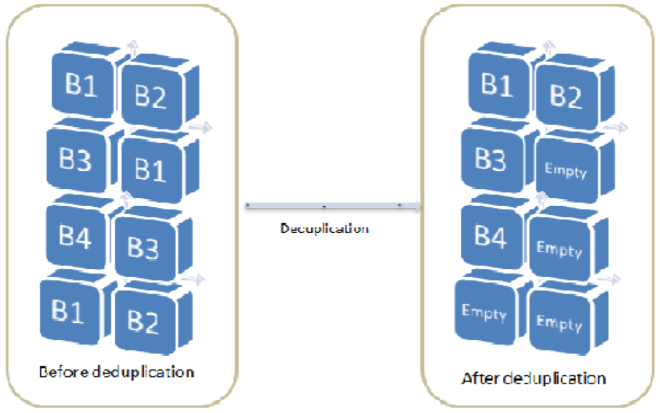

Fig. 1 Data deduplication process

A. Advantages and Disadvantages of Deduplication

R. Kaur et al, 2018 highlighted the major advantages and disadvantages of deduplication techniques.

Advantages

- Decrease storage space.
- Upgrade the network bandwidth
- Saves energy requirement
- Reduction of the overall cost

Disadvantages

- A deduplication technique needs some additional resources.
- Due to hash collision leads to loss of accuracy and consistency of data
- Privacy and security
- Reduction of duplicate copies of data will affect the availability of a storage system.

B. Deduplication Step by Step Procedure

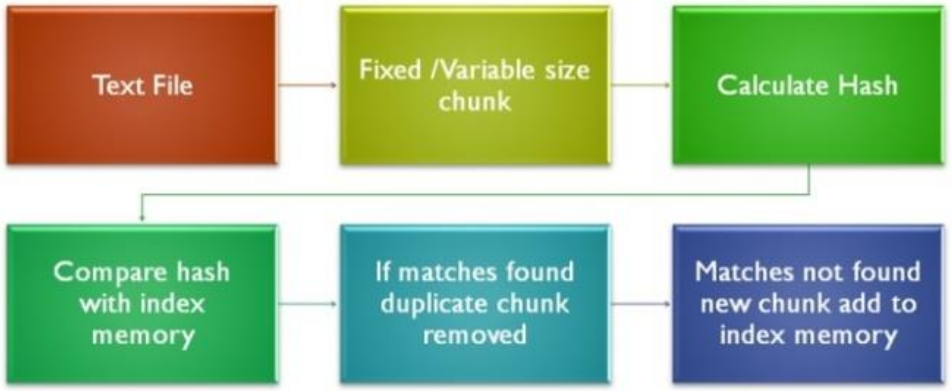
Text File
Fixed /Variable size chunk
Calculate Hash
Compare hash with index memory
If matches found duplicate chunk removed
Matches not found new chunk add to index memory

Fig. 2 Deduplication process

Fig 2 shows the deduplication system workflow of text file divided into fixed or variable size blocks that are called chunking. After chunking for each block, the hash value going to calculated in this process known as fingerprinting. The next step indexing of fingerprints, when user data grows the total size of fingerprints also increased due to this the problem will arise in how to store and manage fingerprints efficiently. Compare the hash value of chunks with the existing one. If matches found duplicate chunk removed else new chunk added to index memory this is also called data compression proposed by Nandakumar, T., 2020. Finally, the stored data need to maintain reliability and security proposed by Draghi, J., 2008.

Categorization of Chunking Algorithm

Chunking processes applied in various disciplines, such as networking proposed by Youn, T.Y, 2018, storage system by Zhou, Y, 2018, data synchronization system, cache system, text recognition, and so on proposed by Lavanya, R, et al., (2017). The chunking process is further classified as: (i) Entire file processing and (ii) Block-wise processing. Entire file processing is means full file is considered as a single block, whereas block processing, the file divided into several blocks. In entire file chunking comparisons made between the files, but in block chunking comparisons made within the file as well as with other files. When dividing a file into blocks or chunks, size can be the fixed size or variable size. The major difficulty is to determine how the chunk or block size should be, either small or large. The chunk or block size is too small it will give better results for the small size of a file, whereas for the huge files this will create performance overhead and indexing problem. If the chunks are large size, it leads to the problem of finding identical ones and reducing the capability of deduplication methods by Youn, T, 2018. So, an optimal chunking algorithm is essential to give better efficiency.

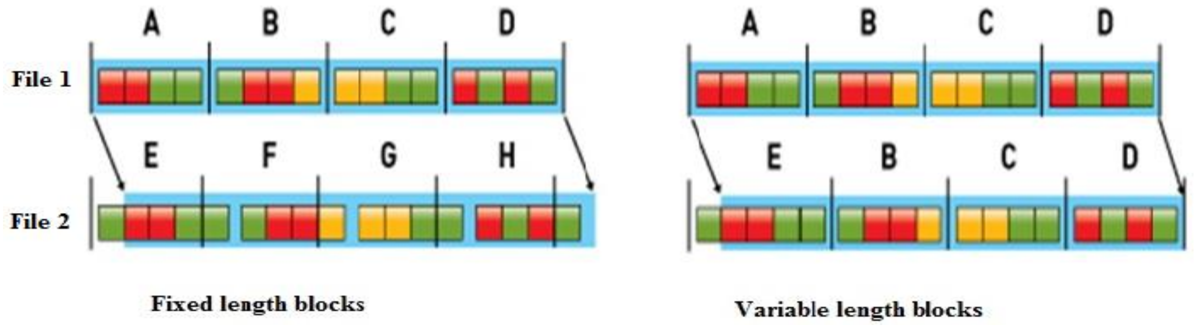

Fig 3. Fixed vs Variable size blocks

Fig 3 shows the fixed and variable size chunking process. Fixed size chunking is very simple and fast, when a byte insert into the file or delete from the file, all the blocks will become completely distinct blocks then it fails to identify duplicate chunks. Fig 3 clearly shows File 1 having A, B, C, and D blocks when inserting a single byte then it is treated as a new file all the chunks are entirely different. So a fixed size chunking process not reliable when inserting or deleting. Content defined chunking (CDC) resolved this issue with the help of variable size chunking. CDC algorithms identify the cut point using the internal features of the file. So that, when byte shifting or insertion only a few chunks are affected, the CDC has better performance of identifying duplicate chunks compared with fixed sized chunking. Variable-length block deduplication gives better granularity control and easy to insert or delete in block. Content defined chunking algorithm permits to use of many methods to find the cut-point like hashes, bytes and neural networks or some machine learning algorithms. When using neural networks and machine learning algorithms gives computational overhead compared with hashes and bytes to find a cut point.

Background

In this section we discussed the background of deduplication and variable size chunking, related work, and their limitations.

A. Deduplication: Background

Deduplication categorizing based on time, level, and location by Zhang, C, 2020. Based on the time it is further classified as inline deduplication and offline deduplication. Inline deduplication needs only less storage compared with offline deduplication because inline deduplication does not store data before processing. Anyway, computation takes time which may be affecting the overall performance. On the other hand, offline deduplication will store data for a short period of time it will lead to some problems when storage is full. Level-based deduplication is classified as byte, block, and file-level. Byte level deduplication methods compared with bytes and remove replicated bytes. When block-level deduplication comparisons are made between blocks or chunks. Location-based deduplication either source-based or target-based, source-based deduplication process done at the client side which means before uploading data to the server. This will decrease the drastic bandwidth cost. Server-based deduplication is done at the server-side, this utilizing more bandwidth.

The fine-grained defragmentation approach used variable size of chunking instead of fixed size, to achieve exact identification of unique data and almost remove fragmented data. The fine-grained defragmentation method not only removed fragmented data and also increases the restore performance by 14 to 329 percentages at the same time reduce the duplicate date by 25 to 87 percent by Hirsch, M, 2016. A systematic deduplication method should satisfy the following criteria (i) Less number of hash calculation, (ii) Minimal amount of hash comparisons, (iii) Best block size selection strategy, and (iv) Fast retrieval.

B. Content Defined Chunking: Background

Fixed-size chunking process lost their efficiency when a byte shifting or inserting, to solve this byte shifting problem content defined variable-length algorithm was proposed by Widodo, R.N, 2017, files are read as a chunks are created using Robin fingerprint. This algorithm is very oldest content defined chunking algorithm. Robin rolling hash is used to find the cut-off point. Cut-off point set on if the hash value of the sliding window fulfills the predefined condition. Earlier Rabin algorithm was the most popular algorithm to calculate the hash value of sliding window in content defined chunking (CDC) by Ponnusamy, V, 2020. Rabin algorithms have computational overhead, time-consuming, and byte shifting problem. Rabin fingerprint has an issue with the size variance of chunks by Karthick, T, 2018. Another CDC algorithm is the Local Maximum Chunking method (LMC) by Anandan, M, 2020. Instead of calculating Robin fingerprint LMC finds the maximum value byte using a sliding window. LMC can overcome byte shifting problem. Slow chunking performance.

To improve the efficiency of LMC, Asymmetric Extremum (AE) algorithm was proposed by Tian, W, 2017. The working principle of AE and LMC algorithm Similar, but AE has two different windows variable-sized window and a fixed window. The number of comparisons significantly reduced and eliminating low entropy strings, more expensive and it's affected by byte shifting problem. A new algorithm Rapid Asymmetric Maximum Algorithm (RAM) was proposed to improve the chunking throughput of AE by Tan, Y, 2017. RAM has low computational overhead and high speed of the chunking process. RAM also reduces the overall cost of the chunking process compared with the AE method.

To enhance the essential characteristics of chunking algorithms a Minimal Incremental Interval (MII) was proposed, it can able to tolerate byte shifting by Zhang, Y, 2015. MII brings the issue of very hard to adjust the average chunk size and efficiency of the algorithm not up to the level. C. Zhang et al, 2015 identifies the drawback of MII and introduced a new algorithm Parity Check of Interval (PCI) can able to tolerate byte shifting problem, and the processing speed PCI is notably less than other algorithms. The drawback of PCI is the increasing transmission compression rate.

C. Motivation

The chunk or block size is too small it will give better results for the small size of a file, whereas for the huge files this will create performance overhead and indexing problems. If the chunks are large size, it leads to the problem of finding identical ones and reducing the capability of deduplication methods proposed by Widodo, R.N, 2017. So, an optimal chunking algorithm is essential to give better efficiency.

1. Performance issue: Mostly offline deduplication method applied in secondary storage, but it needs additional short-term storage space and also raises the I/O bandwidth. There is a need to evolve well organized inline deduplication method with the effective use of resources in a cloud by Saharan, S, 2020.

2. Selecting optimal chunk size: Smaller chunks will save space, but it will generate more hash values. A larger chunk size decreases the hash calculation but it decreases the capacity of the deduplication concept. So, a systematic method needs to calculate the optimum size of chunks.

Exploration of Chunking Algorithm

This section discusses the various content defined chunking algorithms performance and its deficiencies.

A. Rabin based Algorithm

Robin algorithm perceives the cut point done by rolling hash. Rabin fingerprint implemented using polynomial over a finite field. When the window slides old fingerprint can be utilized to compute a new one proposed by Tan, Y, 2018.

The data stream defined as, M.O. Rabin (1981).

$$Rabin(B1, B2, \dots . Bn) = \{\textstyle\sum_{x=1}^{n} B_x\ p^{n-x}\} mod\ D \qquad (1)$$

D= Average chunk size

n = number of bytes in sliding window

B1, B2, ..., Bn = Byte sequence of data stream

Rabin signature calculated by incrementally from previous value as follows:
M.O. Rabin (1981).

$$Rabin(B_{i+1,} B_{i+2,} \dots . . B_{i+n}) = \left\{ \sum_{x=i+1}^{i+n} B_x p^{n-x+i} \right\} mod\ D$$

$$= \left\{ \left[\sum_{x=i}^{i+n-1} B_x p^{n-x+i-1} - B_i p^{n-1} \right] p + B_{i+n} \right\} mod\ D$$

$$\{[Rabin(B_i, B_{i+1}, \dots, B_{i+n-1}) - B_i p^{n-1}] p + B_{i+n}\} mod\ D \qquad (2)$$

Rabin hash having some inadequacy like (i) computational overhead, because of it needs 2 XOR, 1OR, 2 left shifts, and 2 array lookup per byte scanned (ii) Large chunk variance and (iii) Miscalculation of duplicate detection. Fig 4 procedure of Rabin chunking algorithm.

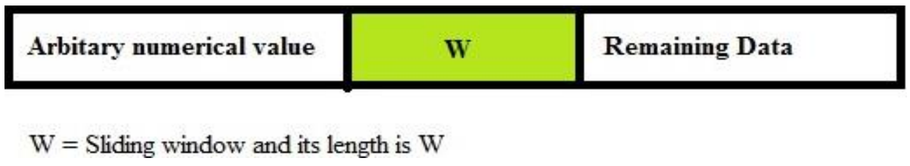

Fig. 4 Operation of Rabin algorithm

B. Local Maximum Chunking Algorithm (LMC)

LMC was introduced to solve the problem of high chunk variance in the Rabin based algorithm proposed by Rabin, M.O, 2018. There is no hash computation required as like Rabin algorithm; LMC used a fixed symmetric window and byte value of files to detect cut points. If the maximum value byte is placed in between two fixed-sized windows, then the cut-off point can be fixed at the end of the data window. If not, the data window moved by one byte again the procedure will continue till end of the data stream proposed by Bjørner, N, 2010. Whenever the data window slides algorithm have to verify all the bytes within the window. So the chunking process will become slow. Computational overhead is intended by time consumption to run the algorithm; the LMC algorithm having four addition, two assignment, and three comparisons. Fig 5 procedure of LMC.

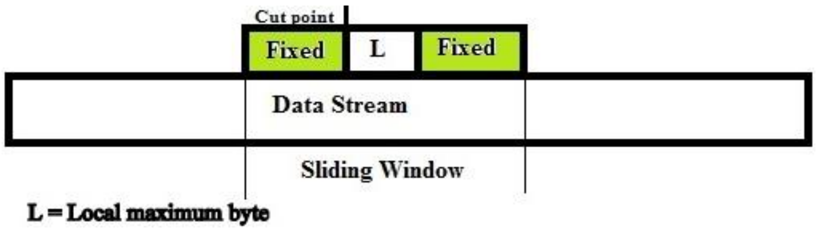

Fig. 5 Operation of LMC algorithm

C. Asymmetric Extremum Algorithm (AE)

AE developed to overcome the drawbacks in LMC and Rabin, 1981. AE gains high performance compared with the existing algorithm, the main reason behind that extreme value in an asymmetric window without having any backtracking process. So AE algorithm very fast and having smaller chunk variance. AE is a similar process to LMC but it has low computational overhead than LMC. AE is done with less comparison to find the cut points.

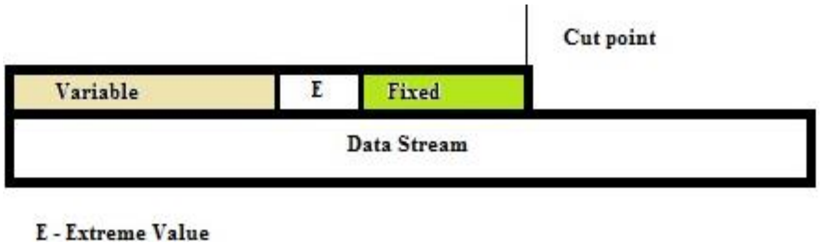

Fig. 6 Operation of AE algorithm

Fig 6. Shows the workflow of the AE algorithm, here extreme value or maximum value must satisfy the condition of extreme value must be greater than the all the bytes in the window left and right, then the cut point decided next byte of the fixed-sized window. In the AE algorithm having some deficiencies, whenever insertion, deletion, or any update on the data stream; byte shifting process will occur AE has less resistance on byte shifting because of the extreme byte value placed in the mid of the chunk.

D. Rapid Asymmetric Maximum Algorithm (RAM)

RAM algorithm used hash less chunking method, using a byte value cut point was identified. RAM also follows the same kind of method of AE to find the cut point, this algorithm too used two windows like variable size and fixed-size window. But the difference is the position of the window placed, at the beginning of the chunk fixed-size window next variable size window placed and then maximum value byte. RAM produced a better result on throughput compared with other chunking algorithms and also useful for client-side deduplication. RAM algorithm having the following comfort,

- Low computational overhead.
- High chunking throughput.
- Fastest chunking speed.
- Reduce the cost of the chunking process.

If the scanned byte greater than the maximum value then a cut point is found. The cut point will be appearing on the right side of the fixed-sized window. The process of the RAM algorithm is shown in Fig 7.

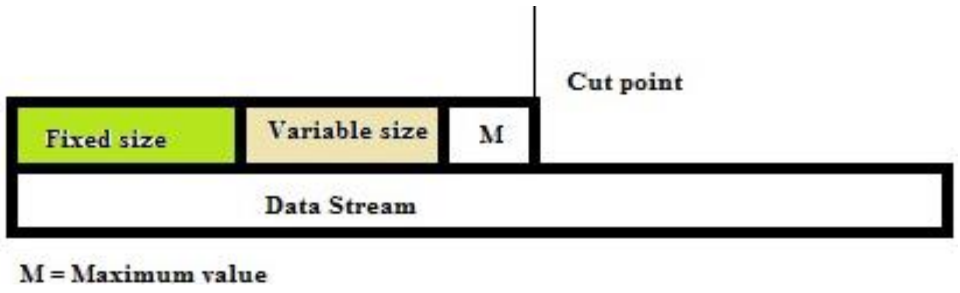

Fig. 7 Operation of RAM algorithm

Chunking Algorithm for Incremental Synchronization

Synchronization techniques are full synchronization and incremental synchronization. Full Synchronization means the full real content is replaced with target data. Incremental synchronization only changed part of data updated with original content. It reduced the cost and time. First, compare target data with real content to identify mismatch content. This section discusses about how MII and PCI chunking algorithm works under incremental synchronization.

A. Minimal Incremental Interval (MII)

MII was developed for incremental synchronization between files. Earlier algorithms (AE, RAM) Chunks are stored in physical disk, so chunk length is considered as the most important factor by Ponnusamy. V, (2020). But in the incremental backup system, the chunking algorithm is only utilized to identifying incremental data; it won't store under the physical disk. So chunk length is not an important factor, for incremental synchronization chunking algorithm fulfils the criteria ability to manage the byte shifting problem.

MII read the data byte by byte; recent read byte compared with the existing byte. If a current byte greater than the existing byte record as a relation < new byte,existing byte>. The number of byte values in the data interval marked as *len*. If the length of the incremental interval reaches preset value then a cut point is found. MII chunking algorithm provides a better ability to resistance against byte shifting, but performance on chunk size variance was very poor, and also the efficiency of an algorithm not good.

B. Parity Check of Interval (PCI)

PCI algorithm overcomes the problem of the MII method. PCI satisfies the condition's byte shifting problem and also can able to locate the changed data exactly in incremental synchronization. Minimize the bandwidth cost of transmission but chunking speed comparatively less than RAM and AE.

PCI read input as a data stream and data window length = 'W', the head point of data window set as the first bye of the file. PCI set two preset value length of data window (W=5) and the threshold of total number of 1s in binary form (NO1BF) V=34. If the data window is filled with NO1BF > the preset value then the cut-off point will be set at end of the data window.

Performance Evaluation

In this section, chunking algorithms are compared with others in terms of chunk characteristics like chunk speed, number of chunks, throughput, and chunk variance. Chunk variance must be less as possible. Table 1 described the dataset. Three dataset generated by using Mersenne Twister Pseudo-Random Number Generator. Because Chunking algorithms performance is also based on the type of dataset, and gave a different set of output for a different type of dataset. For example, the LMC algorithm performed well in the compressed type of file, and the Rabin algorithm works well for network traffic files by Widodo, R.N., et al., (2017). Table 2 illustrated processing time for all chunking algorithms.

Table 1 Dataset

Dataset	**Size**	**Generated by**
Data set 1	2GB	Mersenne Twister Pseudo-Random Number Generator
Data set 2	1.5 GB	Mersenne Twister Pseudo-Random Number Generator
Data set 3	1 GB	Mersenne Twister Pseudo-Random Number Generator

Draghi. J, et al., (2008)

Table 2 Processing Time for all Algorithms

Chunking Algorithm	**Data set 1**	**Data set 2**	**Data set 3**
	Processing time (ms)	**Processing time (ms)**	**Processing time (ms)**
Rabin	2.46	2.11	1.84
LMC	2.61	2.32	2.11
AE	1.90	1.45	1.10
RAM	0.73	0.51	0.33
PCI	2.40	1.90	1.52

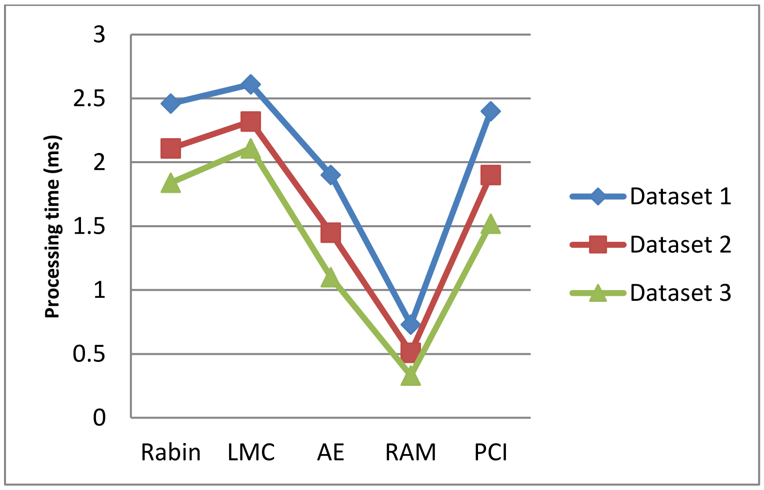

Fig. 8 Processing time of different chunking algorithm

Fig 8 shows chunking speed of different algorithms for three different data set. Among all these algorithms RAM algorithm highest chunking speed and better performance of deduplication. PCI has the best performance in deduplication but it has less processing speed. Table 3 specify the number of chunks in each dataset for all chunking algorithm.

Table 3 Number of Chunks

Algorithms	**Dataset 1**	**Dataset 2**	**Dataset 3**
Rabin	1950	1545	982
LMC	4500	3706	2230
AE	3578	3604	2148
RAM	9500	7000	4500
PCI	9423	6850	4486

Conclusion

In this paper, we have discussed content-defined chunking algorithms for data deduplication and how they overcome the limitations of the fixed-size chunking method. The main drawback of fixed size chunking was the byte shifting problem, when a byte was inserted or deleted at the beginning or middle of the content entire blocks will be changed in this method. Variable size chunking or content defined chunking algorithms works based on the content and also we analysed all the content defined chunking algorithms performance and its limitations. Overall RAM and PCI algorithm performed well in the deduplication process but compared with RAM and PCI, RAM takes less processing time for chunking and it has low computational overhead and high chunking throughput.

References

Kaur, R., Chana, I., & Bhattacharya, J. (2018). Data deduplication techniques for efficient cloud storage management: a systematic review. *The Journal of Supercomputing, 74*(5), 2035-2085. https://doi.org/10.1007/s11227-017-2210-8

Paulo, J., & Pereira, J. (2014). A survey and classification of storage deduplication systems. *ACM Computing Surveys (CSUR), 47*(1), 1-30.

Viji, D., & Revathy, S. (2019). Various Data Deduplication Techniques of Primary Storage. *In International Conference on Communication and Electronics Systems (ICCES),* 322-327.

Nguyen, Q.N., Arifuzzaman, M., Yu, K., & Sato, T. (2018). A context-aware green information-centric networking model for future wireless communications. *IEEE Access, 6,* 22804-22816.

Sanyal, S., & Zhang, P. (2018). Improving quality of data: IoT data aggregation using device to device communications. *IEEE Access, 6,* 67830-67840.

Li, J., Wu, J., & Chen, L. (2018). Block-secure: Blockchain based scheme for secure P2P cloud storage. *Information Sciences, 465,* 219-231. http://www.sciencedirect.com/science/article/pii/S0020025518305012

Youn, T.Y., Chang, K.Y., Rhee, K.H., & Shin, S.U. (2018). Efficient client-side deduplication of encrypted data with public auditing in cloud storage. *IEEE Access, 6,* 26578-26587.

Zhou, Y., Deng, Y., Yang, L.T., Yang, R., & Si, L. (2018). LDFS: A low latency in-line data deduplication file system. *IEEE Access, 6,* 15743-15753.

Zhou, Y., Feng, D., Hua, Y., Xia, W., Fu, M., Huang, F., & Zhang, Y. (2018). A similarity-aware encrypted deduplication scheme with flexible access control in the cloud. *Future Generation Computer Systems, 84*, 177-189.

Lavanya, R., Saranya, P., & Viji, D. (2017). Sampling Selection Strategy for Large Scale Deduplication for Web Data Search. *International Journal of Applied Engineering Research, 12*(11), 2670-2674.

Zhang, C., Qi, D., Li, W., & Guo, J. (2020). Function of Content Defined Chunking Algorithms in Incremental Synchronization. *IEEE Access, 8*, 5316-5330.

Hirsch, M., Ish-Shalom, A., & Klein, S.T. (2016). Optimal partitioning of data chunks in deduplication systems. *Discrete Applied Mathematics, 212*, 104-114.

Widodo, R.N., Lim, H., & Atiquzzaman, M. (2017). A new content-defined chunking algorithm for data deduplication in cloud storage. *Future Generation Computer Systems, 71*, 145-156.

Saharan, S., Somani, G., Gupta, G., Verma, R., Gaur, M.S., & Buyya, R. (2020). QuickDedup: Efficient VM deduplication in cloud computing environments. *Journal of Parallel and Distributed Computing, 139,* 18-31.

Tan, Y., Wang, B., Wen, J., Yan, Z., Jiang, H., & Srisa-an, W. (2018). Improving restore performance in deduplication-based backup systems via a fine-grained defragmentation approach. *IEEE Transactions on Parallel and Distributed Systems, 29*(10), 2254-2267.

Rabin, M.O. (1981). Finger printing by random polynomials, no. TR-15–81. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 15–18.

Bjørner, N., Blass, A., & Gurevich, Y. (2010). Content-dependent chunking for differential compression, the local maximum approach. *Journal of Computer and System Sciences, 76*(3-4), 154-203.

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

Zhang, C., Qi, D., Cai, Z., Huang, W., Wang, X., Li, W., & Guo, J. (2019). MII: A novel content defined chunking algorithm for finding incremental data in data synchronization. *IEEE Access, 7,* 86932-86945.

Tan, Y., & Yan, Z. (2017). Multi-objective metrics to evaluate deduplication approaches. *IEEE Access, 5*, 5366-5377.

Tian, W., Li, R., Xu, Z., & Xiao, W. (2017). Does the content defined chunking really solve the local boundary shift problem?. *In IEEE 36th International Performance Computing and Communications Conference (IPCCC),* 1-8.

Anandan, M., Manikandan, M., & Karthick, T. (2020). Advanced Indoor and Outdoor Navigation System for Blind People Using Raspberry-Pi. *Journal of Internet Technology, 21*(1), 183-195.

Karthick, T., Amith Sai, A.V., Kavitha, P., Jothicharan, J., & Kirthiga Devi, T. (2020). Emotion detection and therapy system using chatbot. *International journal of Advanced Trends in Computer Science and Engineering, 9*(4), 5973 – 5978.

Timande, S., & Dhabliya, D. (2019). Designing multi-cloud server for scalable and secure sharing over web. International Journal of Psychosocial Rehabilitation, 23(5), 835-841.

Nandakumar, T., & Yuvaraj, R. (2020). Data-driven methods for next generation of wireless communication networks. *International Journal of Advanced Trends in Computer Science and Engineering,* 2020, 9(4), 4696–4700.

Dhabliya, D., & Dhabliya, R. (2019). Key characteristics and components of cloud computing. International Journal of Control and Automation, 12(6 Special Issue), 12-18.