

Study of Chunking Algorithm in Data Deduplication

A. Venish and K. Siva Sankar

Abstract Data deduplication is an emerging technology that introduces reduction of storage utilization and an efficient way of handling data replication in the backup environment. In cloud data storage, the deduplication technology plays a major role in the virtual machine framework, data sharing network, and structured and unstructured data handling by social media and, also, disaster recovery. In the deduplication technology, data are broken down into multiple pieces called “chunks” and every chunk is identified with a unique hash identifier. These identifiers are used to compare the chunks with previously stored chunks and verified for duplication. Since the chunking algorithm is the first step involved in getting efficient data deduplication ratio and throughput, it is very important in the deduplication scenario. In this paper, we discuss different chunking models and algorithms with a comparison of their performances.

Keywords Chunking algorithm · Data deduplication · Fixed and variable chunking method

1 Introduction

With the enormous growth of digital data in the cloud storage and enterprise organization, backup and disaster recovery have become crucial to the data center environment. Based on the latest forecast by Cisco, data center traffic on a global scale will grow at a 23 % CAGR, while cloud data center traffic will grow at a faster rate (32 % CAGR) or 3.9-fold growth from 2013 to 2018 [1]. To handle this

A. Venish (✉)

Computer Science and Engineering, Noorul Islam University, Kumarakoil, Tamil Nadu, India
e-mail: Venish07@gmail.com

K. Siva Sankar

Information Technology, Noorul Islam University, Kumarakoil, Tamil Nadu, India
e-mail: sivasankarniu@gmail.com

massive data growth in storage environment and balance the resource consumption, such as RAM/CPU/disk I/O effectively, the data deduplication concept has been accepted worldwide.

There are four major steps involved in data deduplication. The chunking method splits the incoming large data into small portions or ‘chunks’. Using hash algorithm, a unique identifier value is assigned to each chunk. The new incoming chunks are compared with the existing stored chunk using the unique identifier. If the identifier matches, the redundant chunk will be removed and will be given the reference point; otherwise, new chunk will be stored. The key objective of the chunking algorithm is to divide the data object into small fragments. The data object may be a file, a data stream, or some other form of data. There are different chunking algorithms for deduplication including file-level chunking, block-level chunking, content-based chunking, sliding window chunking, and TTTD chunking.

2 Chunking Methods

Efficient chunking is one of the key elements that decide the overall deduplication performance. There are a number of methodologies to detect duplicate chunk of data using fixed-level chunking [2] and fixed-level chunking using rolling checksums [3, 4].

As described by Won et al., chunking is one of the main challenges in the deduplication system [5, 6]. Policroniades et al. [7] and Kulkarni et al. [8] analyzed the various factors that affect the overall deduplication performance using file-level chunking, fixed-size chunking, and content-aware chunking. Also, various studies including Bimodal chunking [9] and Two Threshold Two Denominators (TTTD) [10] emerged to optimize chunking algorithms.

Figure 1 explains the different chunking module structures in detail. The chunking module contains file-level, fixed-size, variable-size, and content-aware chunking. Input files come to the deduplication system and then the files are moved to the chunking module where the file is divided into small chunks based on the chunking method used. After the chunks division, the unique hash value is assigned to each chunk. In this section, we discuss different chunking modules in detail.

2.1 *File-Level Chunking*

File-level chunking or whole file chunking considers an entire file as a chunk, rather than breaking files into multiple chunks. In this method, only one index is created for the complete file and the same is compared with the already stored whole file indexes. As it creates one index for the whole file, this approach stores less number of index values, which in turn saves space and helps store more index values compared to other approaches. It avoids maximum metadata lookup overhead and

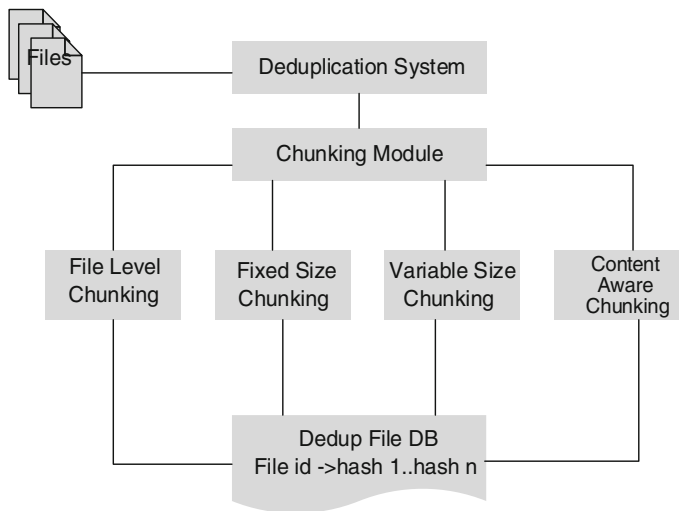


Fig. 1 Different chunking module structure

CPU usage. Also, it reduces the index lookup process as well as the I/O operation for each chunk. However, this method fails when a small portion of the file is changed. Instead of computing the index for the changed parts, it calculates the index for the entire file and moves it to the backup location. Hence, it affects the throughput of the deduplication system. Especially for backup systems and large files that change regularly, this approach is not suitable.

2.2 Block Level Chunking

Fixed-Size Chunking. Fixed-size chunking method splits files into equally sized chunks. The chunk boundaries are based on offsets like 4, 8, 16 kB, etc. [11, 12]. This method effectively solve issues of the file-level chunking method: If a huge file is altered in only a few bytes, only the changed chunks must be reindexed and moved to the backup location. However, this method creates more chunks for larger file which requires extra space to store the metadata and the time for lookup of metadata is more. As it splits the file into fixed size, byte shifting problem occurs for the altered file. If the bytes are inserted or deleted on the file, it changes all subsequent chunk position which results in duplicate index values. Hash collision is likely to happen on chunking method by creating same hash value for different chunks. This can be eliminated by using bit-by-bit comparison which is more accurate, but requires more time to compare the files.

Variable-Size Chunking. The files can be broken into multiple chunks of variable sizes by breaking them up based on the content rather than on the fixed size

of the files. This method resolves the fixed chunk size issue. When working on a fixed chunking algorithm, fixed boundaries are defined on the data based on chunk size which do not alter even when the data are changed. However, in the case of a variable-size algorithm different boundaries are defined, which are based on multiple parameters that can shift when the content is changed or deleted. Hence, only less-chunk boundaries need to be altered. The parameter having the highest effect on the performance is the fingerprinting algorithm.

One of the widely used algorithms for variable-size chunking is Rabin's algorithm [13] to create the chunk boundaries [14–16]. Basically, the variable-size chunking algorithm uses more CPU resources [5, 6]. Based on the characteristics of the file such as content, size, image, color, etc., we can apply variable-size chunking and fixed-size chunking. An ADMAD scheme method proposed by Liu et al. [17] applies not only the fixed or variable chunking method, but is based on the metadata of individual files on which it applies different file chunking methods.

Delta Encoding. Delta encoding is another method to find the boundaries of the file, which prevents repetition of objects relative to each other retaining an existing version of the object with the same name [18]. This method eliminates an object completely in some of the cases, and so the necessary presence of the basic versions, for computing a delta, will be challenging.

Compared to variable-size and fixed-size chunking, Delta encoding gives a good deduplication ratio in desktop applications such as Word, Excel, Zip, Photos, though the problem in delta encoding creates more fingerprints than other methods which yield more space utilization. Bolosky et al. [19] found a file granularity method to detect duplicate data instead of using chunk granularity; on the other hand, Deep Store [20] is planned to work with not only one chunking method but can accommodate various chunking approaches.

Basic Sliding Window. The other approach is the basic sliding window (BSW) approach [21]: it applies break condition logic and marks the boundary of file. Chunk boundary is computed based on the fingerprint algorithm. File boundary is marked based on break condition. The problem with this approach is the chunk size. The size of the chunk cannot be predicted with this approach, but it is possible to predict the probability of getting a larger chunk or a smaller one. This probability is defined based on the probability of getting a particular fingerprint. A divisor D and the sliding window size define if the probability is bigger or smaller.

Two Threshold Two Divisor (TTTD). Another most frequently used variable-length chunking algorithm is TTTD [9, 10, 13, 22]. The Two Threshold Two Divisor (TTTD) chunking method [10] ascertains that chunks smaller than a particular size are not produced. However, it has a drawback that the chunks produced might escape duplicate detection as larger chunks are more likely to be related than smaller ones. To achieve this, the method ignores the chunk boundaries after a minimum size is reached. TTTD applies two techniques where two divisors (D , the regular divisor and D_0 , the backup divisor) are used. By applying these divisors, TTTD guarantees a minimum and maximum chunk size. A minimum and a maximum size limit is used for splitting a file into chunks for searching for duplicates.

TTTD-S algorithm. This is the method proposed by Moh et al. [10] which overcomes the drawback of the TTTD algorithm by introducing TTTD-S algorithm. It computes the average threshold as a new parameter of the main and the backup divisors and uses it as a benchmark for the chunking algorithm. When the average threshold is hit, both primary and secondary divisors shrink to half. The primary and secondary divisors get restored after chunking is completed. This method helps in avoiding a lot of unnecessary computation.

The algorithm by Kruus et al. [9] is based on two fundamental assumptions. The first one is during long runs, where there is a higher probability of getting a large set of unknown data; the second one is during long runs where breaking of large chunks into smaller chunks improves efficiency.

Content or Application Aware-Based Chunking. The chunking methods discussed previously do not consider file format of the file of the underlying data, such as file characteristics. If the chunking method understands the data stream of the file (format of the file), the deduplication method can provide the best redundancy detection ratio compared than the fixed and other variable-size chunking methods. The fixed and the variable-size chunking methods set the chunk boundaries based on the parameter or some predefined condition, whereas the file-type-aware chunking understands the structure of the original file data and defines the boundaries based on the file type such as audio, video, or document.

The advantage of the content-aware chunking method is to facilitate more space savings because this algorithm is aware of the file format and sets the boundaries more natural than other algorithm methods. If the file format is known, it also provides the option to change the chunk size depending on the section of the file. Some formats or file section will not change anytime, by this method it can assume how often the file or section is going to change in the future and the size of the chunk accordingly. For example, potential chunk boundaries include the number of pages in a word document or slides in a PowerPoint presentation.

Douglis et al. [23] used different approach to handle the content-aware deduplication. In this method the data is considered as an object. Incoming data is converted into the object and the same has been compared with the already stored objects for finding the duplicate data in effectively. Using of the Byte level comparison and the knowledge of the content of the data, the input file is split into large data segments. The splitted data segments are compared with the already stored segments and similar segments are determined. At end the altered bytes are saved.

As corroborated by Meister and Brinkmann [24], the understanding of the compression formats such as TAR or ZIP can have consequential space savings. They also exploited the characteristic of the file format and analyzed the deduplication ratio and the scaling efficiency under different chunking methods. They found another interesting thing that playing with Meta data and the payload we can yield more savings space. The knowledge of the archive structure and the matching of the chunk boundaries with the payload boundaries can result in the invalidation of a lot of small size chunks and can save a lot of space by avoiding duplicate data.

However, the mechanism might not be realistic as to grasp all possible file formats and then using them for breaking the files based on their content is a bit

unrealistic for any computational method. Another important issue with file-type chunking is that the algorithm might not be able to detect duplicate data if it comes across in a different file format. This second problem can rather adversely impact the overall performance of this computational approach than improving it. While this approach holds the most promise as far as the space conservation goes, it can utilize more processing power and time if a detailed analysis of all the individual files is done [25, 26].

Figure 2, compares the different deduplicate/undeduplicate size with different chunk size algorithm. File-level and fixed-size chunking occupying more spaces compare than the Rabin chunking algorithm. Rabin chunking algorithm provides good results for all the chunk size such as 8, 16, 32, and 64 kB [27]. Figure 3 compares the different deduplicate/undeduplicate sizes with different deduplication file system (Domain Size) size. As the domain size increases, amount of space reclaimed by the deduplication algorithm raises linearly. Best possible result is achieved in 8 kB Rabin algorithm. While using whole file without chunking yields to the worst deduplication result [27].

Fig. 2 Comparison of different deduplication versus various chunk size

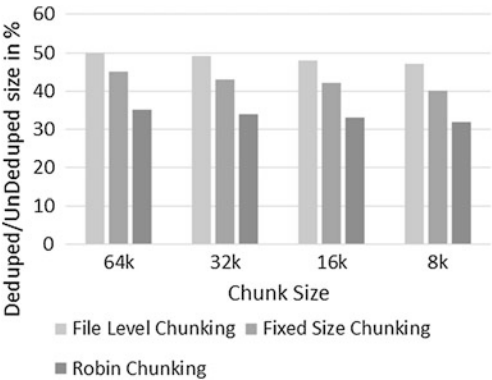
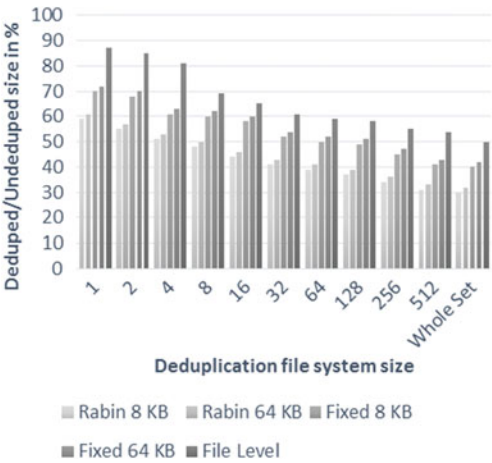


Fig. 3 Deduplication versus deduplication domain size



3 Conclusion

We have summed up different algorithms and methods for the deduplication of data. In the data deduplication process, the first and the most important step is the granular division and subdivision of data. For proper granularity, an effective and efficient chunking algorithm is a must. If the data is chunked accurately, it increases the throughput and the net deduplication performance. The file-level chunking method is efficient for small files deduplication, but not relevant for a big file environment or a backup environment. The problem with the fixed-size chunking method is that it fails to detect redundant data if some bytes are altered. The variable-size chunking methods overcome this fixed file size chunking problem by creating boundaries based on the content of the files. In the second section, we discussed various variable chunking methods. In the current deduplication scenario, the content-based chunking methods provide good throughput as well as decrease the space utilization. While working and managing the multimedia files like images, videos, or audio, the content-aware chunking methods are accepted mostly.

References

1. Cisco Global Cloud Index: Forecast and methodology (2015) white paper. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.html. Visited last on 02 Apr 2015
2. Quinlan S, Venti SD (2002) A new approach to archival storage. In: Proceedings of the first USENIX conference on file and storage technologies, Monterey, CA
3. Denehy TE, Hsu WW (2003) Reliable and efficient storage of reference data. Technical Report RJ10305, IBM Research, Oct 2003
4. Andrew Tridgell (1999) Efficient algorithms for sorting and synchronization. PhD thesis, Australian National University
5. Won Y, Kim R, Ban J, Hur J, Oh S, Lee J (2008) Prun: eliminating information redundancy for large scale data backup system. In: Proceedings IEEE international conference computational sciences and its applications (ICCSA'08)
6. Won Y, Ban J, Min J, Hur J, Oh S, Lee J (2008) Efficient index lookup for de-duplication backup system. In: Proceedings of IEEE international symposium modeling, analysis and simulation of computers and telecommunication systems (MASCOTS'08), pp 1–3, Sept 2008
7. Kulkarni P, Douglass F, LaVoie J, Tracey J (2004) Redundancy elimination within large collections of files. In: Proceedings of the USENIX annual technical conference, pp 59–72
8. Kruus E, Ungureanu C, Dubnicki C (2010) Bimodal content defined chunking for backup streams. In: Proceedings of the 8th USENIX conference on file and storage technologies. USENIX Association
9. Policroniades C, Pratt I (2004) Alternatives for detecting redundancy in storage systems data. In: Proceedings of the annual conference on USENIX annual technical conference. USENIX Association
10. Eshghi K, Tang HK (2005) A framework for analyzing and improving content-based chunking algorithms
11. Kubiawicz J et al (2000) Oceanstore: an architecture for global store persistent storage. In: Proceedings of the 9th international conference on architectural support for programming languages and operating systems

12. Quinlan S, Dorwards S (2002) Venti: a new approach to archival storage. In: Proceedings of USENIX conference on file and storage technologies
13. Rabin M (1981) Fingerprinting by random polynomials. Center for Research in Computing Technology, Aiken Computation Laboratory, University
14. Lillibridge M, Eshghi K, Bhagwat D, Deolalikar V, Trezise G, Camble P (2009) Sparse indexing: large scale, inline deduplication using sampling and locality. In: Proceedings of the 7th USENIX conference on file and storage technologies (FAST'09), San Francisco, CA, USA, Feb 2009, pp 111–124
15. Muthitacharoen A, Chen B, Mazières D (2001) A low-bandwidth network file system. SIGOPS Oper Syst Rev 35(5):174–187
16. Zhu B, Li K, Patterson H (2008) Avoiding the disk bottleneck in the data domain deduplication file system. In: FAST'08: Proceedings of the 6th USENIX conference on file and storage technologies, Berkeley, CA, USA, pp 1–14
17. Liu C, Lu Y, Shi C, Lu G, Du D, Wang D (2008) ADMAD: application-driven metadata aware de-duplication archival storage system. In: Proceedings of fifth IEEE international workshop storage network architecture and parallel I/Os (SNAPI'08), pp 29–35
18. Mogul J, Douglass F, Feldmann A, Krishnamurthy B (1997) Potential benefits of delta encoding and data compression for HTTP. In: Proceedings of ACM SIGCOMM'97 conference, pp 181–194, Sept 1997
19. Bolosky WJ, Corbin S, Goebel D, Douceur JR (2000) Single instance storage in windows 2000. In: Proceedings of fourth USENIX windows systems Symposium, pp 13–24
20. You LL, Pollack KT, Long DDE (2005) Deep store: an archival storage system architecture. In: Proceedings of international conference on data engineering (ICDE'05), pp 804–8015
21. Muthitacharoen A, Chen B, Mazieres D (2001) A low-bandwidth network file system. ACM SIGOPS Oper Syst Rev 35(5):174–187
22. Thein NL, Thwel TT (2012) An efficient Indexing Mechanism for data de-duplication. In: Proceedings of the 2009 international conference on the current trends in information technology (CTIT), pp 1–5
23. Bloom BH (1970) Space/time tradeoffs in hash coding with allowable errors. Commun ACM 13(7):422–426
24. Meister D, Brinkmann A (2009) Multi-level comparison of data deduplication in a backup scenario. In: Proceedings of SYSTOR'09: The Israeli experimental systems conference, May 2009, pp 1–12
25. Cannon D (2009) Data deduplication and tivoli storage manager, Mar 2009
26. Data Domain LLC. Deduplication FAQ. url:<http://www.datadomain.com/resources/faq.html>
27. Meyer DT, Bolosky WJ (2011) A study of practical deduplication. In: Proceedings of 9th USENIX conference on file and storage technologies

Proceedings of the International Conference on Soft
Computing Systems

ICSCS 2015, Volume 2

Suresh, L.P.; Panigrahi, B.K. (Eds.)

2016, XV, 743 p. 343 illus. in color., Softcover

ISBN: 978-81-322-2672-7