ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 6, July 2017

Effective Big Data Storage Framework for Cloud Memory Management

Jaai Arankalle, Shubhangi Bagade, Saee Joshi, Rutuja Limaye

U.G. Student, Department of Computer Engineering, Siddhant College of Engineering, Pune, Maharashtra, India

ABSTRACT: Big sensing data is extensively used in both industry and scientific research applications where the data is generated with huge volume. Cloud computing provides a best platform for big sensing data processing and storage. It moves around four important areas of analytics and Big Data, namely (i) data management and supporting architectures; (ii) model development and scoring;(iii) visualization and user interaction; and (iv) business models. However, the storage pressure on cloud storage system caused by the explosive growth of data is growing by the day, especially a vast amount of redundant data waste a lot of storage space. Data deduplication can effectively reduce the size of data by eliminating redundant data in storage systems. In cloud data storage, the deduplication technology plays a major role. In the deduplication technology, data are broken down into multiple pieces called “chunks”. The Two Thresholds Two Divisors (TTTD) algorithm is used for chunking mechanism and is used for controlling the variations of the chunk-size.

KEYWORDS:cloud computing, data chunk, data compression, big sensing data, scalability.

I. INTRODUCTION

It is becoming a big necessity that we need to process big data from multiple sensing systems. Cloud storage systems are able to give low-cost, convenient and good network storage service for users, which makes them more popular. However, the storage pressure on cloud storage system caused by the huge growth of data is growing by the day, especially a vast amount of repetitive waste plenty of storage space. Data deduplication can operatively reduce the size of data by excluding repetitive data in storage systems. However, current researches on data deduplication, which mainly concentrate on the static scenes such as the backup and archive systems, are not suitable for cloud storage system due to the dynamic nature of data[4]. Deduplication applied in cloud storage systems can minimize the size of data and save the network bandwidth, the dynamicity of data in cloud storage systems are different from backup and archive systems, which brings different approaches for the study of data deduplication in cloud storage systems. Here, the dynamic characteristics of data are caused by dynamic sharing between multiple users. For example, the same data accessed by different users and the access frequency of different data at the same time is different, the access frequency of the same data changes overtime and duplicated data appears again in different (storage nodes)nodes for data modification by users[6].There are many different deduplication approaches depending on the range of deduplication (locally or globally), the position of deduplication (at the client or server side), the time of deduplication (inline or offline), and the granularity of deduplication (file-level or chunk-level). The process of deduplication mainly comprises four steps: (1) chunking; (2) calculating fingerprint; (3) fingerprint lookup (finding out the redundancy by fingerprint comparison); storing new data[6]. Chunking can break a file into small parts called chunks for detecting more redundancy. There are several typical chunking strategies of data deduplication [6], such as whole-file chunking, fixed-size chunking, content-defined chunking, and Two Thresholds Two Divisors(TTTD) chunking.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 6, July 2017

II. SYSTEM MODEL

A. **REGISTRATION AND AUTHENTICATION:**Registration page consist of details as follows:

i) First name
ii) Last name
iii) Gender
iv) Mobile number
v) Email id
vi) Verify email
vii) Send message

Authentication is done through email id. First user has to register and create id and verified email does the work of checking whether the user is registered or not. When user will send message to server and if the user is registered the message will be successfully sent to server. For security purpose the server will send password to user's mail after registration.

B. UPLOADER & DOWNLOADER:

In the uploader module, first we will upload the file which we want to compress. The compressed file will be uploaded on cloud as well as saved on client side. If the client machine gets corrupt then the cop of compressed file will be available on cloud.

The downloader module does the work of downloading the file stored on cloud. The uploaded file which is in a compressed form and at the time of downloading the file will be in a decompressed form which will be made available to user in its original form.

C. COMPRESSION MODULE:

In compression module, the file which is to be uploaded on cloud is first divided into chunks using TTTD algorithm by using the following formula:

Chunk size= (length % number of cores == 0 ? length % number of cores | length / number of cores + 1).

Where, length is the size of file

These compressed chunks will be integrated and the whole file will be uploaded on cloud.

D. **DECOMPRESSION MODULE:**The uploaded file on cloud if it is requested by the user then the user can download it and it will be available in a decompressed form.

E. **PERFORMANCE ANALYZER:**Performance Analyzer gives the comparison on the basis of time and memory for the file before compression and after compression.

III. OPPORTUNITIES FOR COMPRESSION

The data accessed by real-world applications show significant amount of redundancy which provides opportunity for compression. Such redundancy may arise due to use of constants, nature of program inputs and operations such as assignment and copying. For example, several applications use a common value for initializing a large array and in an image processing application, multiple adjacent pixels may have the same color. Similarly, on using memcpy() two copies of data may be stored in the cache. Due to these factors, some benchmarks can contain up to 98 percent of duplicated blocks. Further, it has been shown that if each unique memory value is stored exactly once in the cache, compression ratios beyond 16_ can be achieved. However, due to practical considerations (discussed below), most practical techniques achieve compression ratios of less than 2_, although a few techniques achieve compression ratios of 3-4_. In several cases, programmers provision large size data types to handle worst-case scenario, while most values can fit in a smaller data type, for example, a four-byte integer may store only a one-byte value. Such values, referred to as narrow-width values, can be easily compressed. Similarly, 'special patterns', such as data values where all the bits are either zero or one, can be coded using special flags or smaller number of bits. Further, in many cases, the

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 6, July 2017

differences between values stored within the cache line may be small and hence, they can be represented in a compressed form using a base value and an array of differences.[2]

A. **FACTORS AND TRADEOFFS:**

The efficacy of compression approach depends on several factors. In what follows, we discuss some of these factors. Compressibility of Data. Different applications and data have different compressibility, which can greatly affect the improvements obtained from the compression approach. In case of incompressible data or inefficient compression algorithm, the size of a compressed block may even be larger than that of uncompressed block due to use of metadata etc. For such cases, storing data in uncompressed form is beneficial. For example, compression approach exploits data redundancy, and hence, it may not benefit systems which use encrypted data since data encryption techniques introduce randomness and reduce data redundancy Larger block sizes are expected to compress well due to higher redundancy, however, at large sizes, access to even a single sub block will require decompressing the whole block. On the other hand, for some compression algorithms such as zero-content (ZC) detection, use of smaller blocks may lead to larger compression ratios since finding certain patterns may be easier. However, use of smaller block sizes may lead to larger overhead of metadata.

B. **TABLE OF FACTOR :**

Table 1
A classification of a research work

Classification	References
Processor Component	
Memory compression	[7], [8], [9]
Compression algorithms used/evaluated	
LZ and variants	[9],[2], [11]
Huffman coding	[5],[12]
Run length encoding	[9]
Shannon Fano	[9]
Arithmetic encoding	[9]
TTTD	[5]
Map Reduce	[1]
Objective	
Performance improvement	[7], [2], [6]
Energy saving	[2], [9], [5]
Capacity increase and miss rate reduction	[2], [1]
Bandwidth reduction	[6],
Interaction with other approaches and use in other contexts	
Prefetching	[2]
Resource-scaling and energy saving	[4], [1], [5]
Error-correction	[3]
Link compression	[1]

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

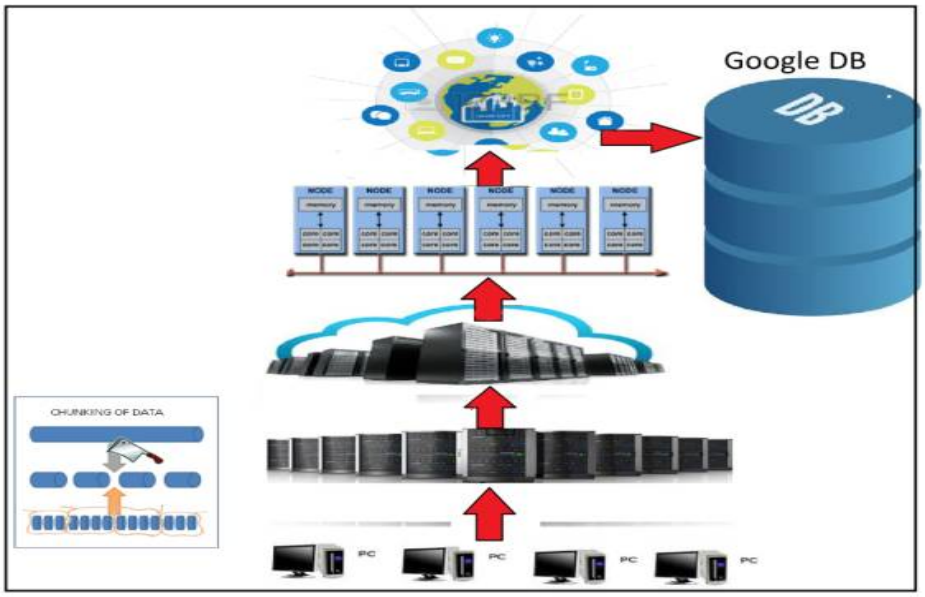
International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 6, July 2017

IV. SYSTEM ARCHITECTURE



For fast processing data is divided into chunks or segments. Chunking mechanism is used to divide the data into small parts at the server side so that less time will be used for uploading the data .Chunking is use to increase faster processing for evaluating frequent data transmission on cloud . Main problem is time and space management so to reduce this memory optimization Dynamic load balancing is applied.

V. ALGORITHM

A. **TTTD ALGORITHM :**The maximum and minimum thresholds are used to eliminate very large-sized and very small-sized chunks in order to control the variations of chunk-size. The main divisor plays the same role as the BSW algorithm and can be used to make the chunk-size close to our expected chunk-size. In usual, the value of the second divisor is half of the main divisor. Due to its higher probability, second divisor assists algorithm to determine a backup breakpoint for chunks in case the algorithm cannot find any breakpoint by main divisor. According to the research, we list these four parameters and their optimal values when considering the expected chunk-size is 1000 bytes.[5]

(1) The algorithm shifts one byte at one time and computes the hash value.

(2) If the size from last breakpoint to current position is larger than minimum threshold, it starts to determine the breakpoint by second and main divisors.

(3) Before the algorithm reaches the maximum threshold, if it can find a breakpoint by main divisor, then uses it as the chunk boundary. The sliding window starts at this position and repeats the computation and comparison until the end of file.

(4) When the algorithm reaches the maximum threshold, it uses the backup breakpoint if it found any one, otherwise use the maximum threshold as a breakpoint.

B. **HUFFMAN ALGORITHM :** Huffman Encoding Algorithms use the probability distribution of the alphabet of the source to develop the code words for symbols. The frequency distribution of all the characters of the source is calculated in order to calculate the probability distribution. According to the probabilities, the code words are assigned. Shorter code words for higher probabilities and longer code words for smaller probabilities are assigned. For this task a binary tree is created using the symbols as leaves according to their probabilities and paths of those are taken as the code words. Two families of Huffman Encoding have been proposed: Static Huffman Algorithms and Adaptive

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 6, July 2017

Huffman Algorithms. Static Huffman Algorithms calculate the frequencies first and then generate a common tree for both the compression and decompression processes. Details of this tree should be saved or transferred with the compressed file. The Adaptive Huffman algorithms develop the tree while calculating the frequencies and there will be two trees in both the processes. In this approach, a tree is generated with the flag symbol in the beginning and is updated as the next symbol is read.

VI. EXPERIMENTAL RESULT

A. **DATASET :**

Serial no:	Document	Extension	Size
1.	text	.txt	4KB
2.	PDF	.pdf	184KB
3.	Application	.exe	1.5MB
4.	MS Word	.doc	35KB
5.	MS Excel	.xls	97KB
6.	Latex	.tex	15KB
7.	Image	.jpeg	25KB
8.	Audio	.mp3	10MB
9.	Video	.mp4	51MB

B. **COMPUTATIONAL NODE :**

	Node 1 (Server)	Node 2 (Client 1)	Node 3 (Client 3)
Processor	I3	I5	Pentium
RAM	8 GB	4 GB	2 GB
Speed	2.0GHz	2.0GHz	1.0GHz
Java	7	7	7
OS	Windows 10	Windows 10	Windows 7
Hard disk	1 TB	1 TB	500

C. **RESULT:**

	Upload		Download	
	Normal	Proposed	Normal	Proposed
DB 1	10	8	10	10
DB 2	20	16	20	20
DB 3	30	24	30	30

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

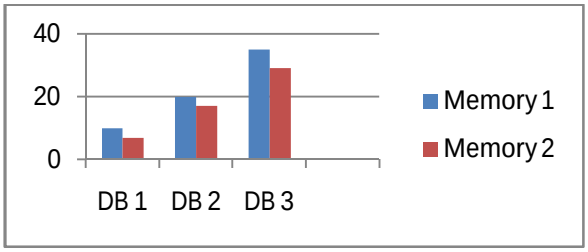
International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Special Issue 6, July 2017

D. **GRAPH:**



VII. CONCLUSION

In this paper, we have completed a novel scalable data compression based on similarity calculation among the partitioned data chunks with Cloud computing. 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. TTTD-S algorithm, not only successfully achieves the significant improvements in running time and average chunk-size, but also obtains the better controls on the variations of chunk-size by reducing the large-sized chunks.

REFERENCES

1. .Chi Yang, Jinjun Chen, “A Scalable Data Chunk Similarity based Compression Approach for Efficient Big Sensing Data Processing Cloud”, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, February 2016.
2. Sparsh Mittal, Member, IEEE and Jeffrey S. Vetter, Senior Member, IEEE ”A Survey Of Architectural Approaches for Data Compression in Cache and Main Memory Systems” IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 27, NO. 5, MAY 2016.
3. Santoshi Tsuchiya, Yoshinori Sakamoto, Yuichi Tsuchimoto, Vivian Lee, “Big Data Processing on Cloud Environment”, FUJITSU Science and Technology Journal, 48(2):159-168, 2012.
4. Venish and K. Siva Sankar, “Study of chunking algorithm in Data Deduplication”, Proceedings of the International Conference on Soft Computing Systems, Advances in Intelligent Systems and Computing 398, DOI 10.1007/978-81-322-2674-1_2.
5. Chang, BingChun, "A Running Time Improvement for Two Thresholds Two Divisors Algorithm" (2009).*Master's Projects*. Paper 42.
6. Xiaolong Xu, Qun Tu, “Data deduplication mechanism for cloud storage systems”, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING VOL:25 NO:7 SEPTEMBER, 2015
7. Marcos D. Assunção a, Rodrigo N. Calheiros b, Silvia Bianchi c, “ Big data computing and clouds : Trends and future directions”,Journal of parallel and distributed computing,79-80(2015) 3-15
8. Amit Jain, a Kamaljit I. Lakhtariab, Prateek Srivastava,” A Comparative Study of Lossless Compression Algorithm on Text Data”,Proc. of Int. Conf. on Advances in Computer Science, AETACS
9. S.R. KODITUWAKKU,”COMPARISON OF LOSSLESS DATA COMPRESSION ALGORITHMS FOR TEXT DATA”, S.R. Kodituwakku et. al. / Indian Journal of Computer Science and Engineering Vol 1 No 4 416-425.
10. K. Tanaka and A. Matsuda, “Static energy reduction in cache memories using data compression,” in Proc. IEEE TENCON, 2006,pp. 1–4.
11. S. Roy, R. Kumar, and M. Prvulovic, “Improving system performance with compressed memory,” in Proc. Int. Parallel Distrib. Process. Symp., 2001, pp. 7–13.
12. J.-S. Lee, W.-K. Hong, and S.-D. Kim, “Design and evaluation of a selective compressed memory system,” in Proc. Int. Conf. Comput. Des., 1999, pp. 184–191.