ISSN 2395-1621

Intelligent Backup System

[#1]Rashmi Mandave, [#2]Shrinivas Dube, [#3]Vaibhav Chhajed, [#4]Pratik Bobde,
[#5]Mrs. Shikha Pachouly

[1]mandaverashmi1994@gmail.com
[2]shrinivasdube16@gmail.com
[3]vaibhavchhajed67@gmail.com
[4]pratbob38@gmail.com
[5]sjpachouly@aissmscoe.com

[#1234]Department of Computer Engineering, Savitribai Phule Pune University
AISSMS COE, Pune, India

.

ABSTRACT

In personal computing devices that rely on a cloud storage environment for data backup, an imminent challenge facing source deduplication for cloud backup services is the low deduplication efficiency due to a combination of the resource intensive nature of deduplication and the limited system resources. An Application-aware data deduplication scheme that improves data deduplication efficiency by exploiting application awareness, and further combines local and global duplicate detection to strike a good balance between cloud storage capacity saving and deduplication time reduction. Our scheme can significantly improve deduplication efficiency over the state-of-the-art methods with low system overhead, resulting in shortened backup window, increased power efficiency and reduced cost for cloud backup services of personal storage.

Keywords—Private storage, cloud backup, deduplication efficiency, application awareness.

ARTICLE INFO

Article History

Received :5th February 2016

Received in revised form :

6th February 2016

Accepted :7th February , 2016

Published online :

9th February 2016

I. INTRODUCTION

In this system, data deduplication achieved at client side as well cloud side and further combines both to achieve good balance between storage capacity and reduction time. The storage service provider has to store data in the form of chunks, it is first generates fingerprints by using hash functions and transferred over WAN through internet connectivity.

In cloud computing, data deduplication is a specialized data compression technique for eliminating duplicate copies of repeating data. Related and somewhat synonymous terms are intelligent (data) compression and single-instance (data) storage. This technique is used to improve storage utilization and can also be applied to network data transfers to reduce the number of bytes that must be sent. In the deduplication process, unique chunks of data, or byte patterns, are identified and stored during a process of analysis. As the analysis continues, other chunks are compared to the stored copy and whenever a match occurs, the redundant chunk is replaced with a small reference that points to the stored chunk. Given that the same byte pattern may occur dozens, hundreds, or even thousands of times

(the match frequency is dependent on the chunk size), the amount of data that must be stored or transferred can be greatly reduced. Deduplication may occur "in-line", as data is flowing, or "post-process" after it has been written.

A. Post-process deduplication

With post-process deduplication, new data is first stored on the storage device and then a process at a later time will analyse the data looking for duplication. The benefit is that there is no need to wait for the hash calculations and lookup to be completed before storing the data thereby ensuring that store performance is not degraded. Implementations off ering policy-based operation can give users the ability to defer optimization on "active" files, or to process files based on type and location. One potential drawback is that you

may unnecessarily store duplicate data for a short time which is an issue if the storage system is near full capacity.

B. In-line deduplication

This is the process where the deduplication hash calculations are created on the target device as the data enters the device in real time. If the device spots a block that it already stored on the system it does not store the new block, just references to the existing block. The benefit of in-line deduplication over post process deduplication is that it requires less storage as data is not duplicated. On the negative side, it is frequently argued that because hash calculations and lookups takes so long, it can mean that the data ingestion can be slower thereby reducing the backup throughput of the device. However, certain vendors with inline deduplication have demonstrated equipment with similar performance to their post-process deduplication counterparts.

C. Source versus target deduplication

Another way to think about data deduplication is by where it occurs. When the deduplication occurs close to where data is created, it is often referred to as "source deduplication". When it occurs near where the data is stored, it is commonly called "target deduplication".

- Source deduplication ensures that data on the data source is deduplicated. This generally takes place directly within a file system. The file system will periodically scan new files creating hashes and compare them to hashes of existing files. When files with same hashes are found then the file copy is removed and the new file points to the old file. Unlike hard links however, duplicated files are considered to be separate entities and if one of the duplicated files is later modified, then using a system called copy-on-write a copy of that file or changed block is created. The deduplication process is transparent to the users and backup applications. Backing up a deduplicated file system will often cause duplication to occur resulting in the backups being bigger than the source data.

- Target deduplication is the process of removing duplicates when the data was not generated at that location. Example of this would be a server connected to a SAN/NAS, The SAN/NAS would be a target for the server (Target deduplication). The server is not aware of any deduplication, the server is also the point of data generation.

II. ARCHITECTURAL OVERVIEW

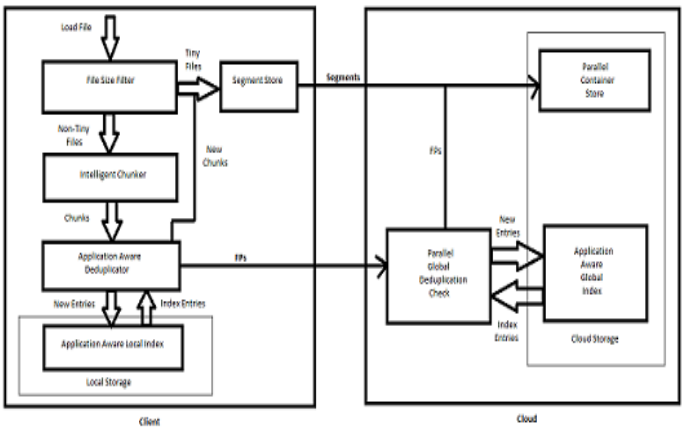

Fig 1. Architecture

An architectural overview is illustrated in Fig. 1, where tiny files are first filtered out by file size filter for efficiency reasons, and backup data streams are broken into chunks by an intelligent chunker using an application aware chunking strategy. Data chunks from the same type of files are then deduplicated in the application-aware deduplicator by generating chunk fingerprints in hash engine and performing data redundancy check in application-aware indices in both local client and remote cloud. Their fingerprints are first looked up in an application aware local index that is stored in the local disk for local redundancy check. If a match is found, the metadata for the file containing that chunk is updated to point to the location of the existing chunk. When there is no match, the fingerprint will be sent to the cloud for further parallel global duplication check on an application aware global index, and then if a match is found in the cloud, the corresponding file metadata is updated for duplicate chunks, or else the chunk is new. On the client side, fingerprints will be transferred in batch and new data chunks will be packed into large units called segments in the segment store module with tiny files before their transfers to reduce cloud computing latency and improve network bandwidth efficiency over WAN. On the cloud data center side, segments and its corresponding chunk fingerprints are stored in a self-describing data structure container in cloud storage, supported by the parallel container store.

III. MODULES

A. Login Module

In this module, we take username and password from user to authenticate login to application.

B. File Upload Module

In this module, we read file from client and save it at local disk space.

C. File filter Module

This module, checks if uploaded file is less than the defined chunk sizes. If less it won't send it for chunking and put them in different segment store. Once a segment is full, the file is chunked.

D. Chunking Module

This module has 2 sub modules:

- Static chunking:

This module chunks all the files that are non-editable like file with extensions .exe, .pdf, .jpg etc.

- Dynamic chunking:

This module chunks all the files that are editable like files with extensions .doc, .txt, .xls, .docx etc.

E. Fingerprinting Module

This module takes care of generating fingerprints for all the chunks generated by chunking module depending upon the type of chunking techniques used. If static chunking has been applied then SHA1 is used for fingerprinting. If dynamic chunking has been applied then MD5 is used for fingerprinting. Once fingerprints are generated they will be updated in local index.

NOTE: Fingerprinting here means generating hash codes for each chunk that are unique to the chunk.

F. Synchronisation Module

This module takes care of synchronizing the local index file and global Index file. We send the updated portions of local index file to global index file so that they are in sync.

G. Chunk Transfer Module

This module includes all activities that take care of sending chunks from local storage to global storage.

H. Chunk Receive Module

This module includes all activities that take care of receiving chunks from local storage at global storage.

I. Update Chunk Module

This module includes all activities that take care of updating received chunks at global storage.

J. Delete File Module

This module includes all activities regarding deleting a file at local storage and its activities regarding synchronization with global storage.

K. Download File Module

This module includes all activities that take care of downloading a file stored at global storage.

L. Logout Module

In this module we destroy session of logged user from application.

IV. ALGORITHMS

A. Message Digest 5:

It is a widely used cryptographic function with a 128- bit hash value. MD5 has been employed in a wide variety of security applications, and is also commonly used to check the integrity of files. An MD5 hash is typically expressed as a 32 digit hexadecimal number. MD5 processes a variable-length message into a fixed-length output of 128 bits.

Steps:

a. The input message is broken up into chunks of 512-bit blocks (sixteen 32-bit little endian integers), the message is padded so that its length is divisible by 512.

b. The padding works as follows: first a single bit, 1, is appended to the end of the message.

c. This is followed by as many zeros as are required to bring the length of the message up to 64 bits less than a multiple of 512.

d. The remaining bits are filled up with a 64-bit integer representing the length of the original message, in bits.

e. The MD5 algorithm uses 4 state variables, each of which is a 32 bit integer (an unsigned long on most systems). These variables are sliced and diced and are (eventually) the message digest. The variables are initialized as follows:
A = 0x67452301
B = 0xEFCDAB89
C = 0x98BADCFE
D = 0x10325476

f. Now on to the actual meat of the algorithm: the main part of the algorithm uses four functions to thoroughly goober the above state variables.

g. These functions, using the state variables and the message as input, are used to transform the state variables from their initial state into what will become the message digest. For each 512 bits of the message, the rounds performed after this step, the message digest is stored in the state variables. To get it into the hexadecimal form you are used to seeing, output the hex values of each the state variables, least significant byte first. For example, if after the digest:
A = 0x01234567
B = 0x89ABCDEF
C = 0x1337D00D
D = 0xA5510101

Then the message digest would be:
67452301EFCDAB890DD03713010151A5
(required hash value of the input value).

B. Secure Hash Algorithm 1:

The Secure Hash Algorithm is the original 160-bit hash function resembling the earlier MD5 algorithm. The SHA1 algorithm uses 5 state variables, each of which is a 32 bit

integer (an unsigned long on most systems). These variables are sliced and diced and are (eventually) the message digest. The variables are initialized as follows:
h0 = 0x67452301
h1 = 0xEFCDAB89
h2 = 0x98BADCFE
h3 = 0x10325476
h4 = 0xC3D2E1F0

There are 80 rounds in SHA1 Algorithm.
The hash value generated by the sha1 hash function.

SHA1 ("The quick brown fox jumps over the lazy dog")
= 2fd4e1c6 7a2d28fc ed849ee1 bb76e739 1b93eb12.

Even a small change in the message will, with overwhelming probability, result in a completely diff erent hash due to the avalanche eff ect. For example, changing Dog to cog produces a hash with diff erent values for 81 of the 160 bits:

SHA1 ("The quick brown fox jumps over the lazy cog")
= de9f2c7f d25e1b3a fad3e85a 0bd17d9b 100db4b3.

V. CONCLUSION

Our application, intelligent backup system improves deduplication efficiency by combining both local and global deduplication and minimizes computational overhead by making adaptive use of hash functions. Also, improves data transfer efficiency using a data aggregation strategy. Thus, our application can be used to efficiently backup private/personal storage to cloud storage.

ACKNOWLEDGEMENT

Apart from our own, the success of this paper depends largely on the encouragement and guidelines of many others. We are especially grateful to our guide Prof S.J.Pachouly and Prof D. P. Gaikwad, Head of Computer Engineering Department, AISSMSCOE who has provided guidance, expertise and encouragement. We are thankful to the staff of Computer Engineering Department for their cooperation and support .We would like to put forward our heartfelt acknowledgement to all our classmates, friends and all those who have directly or indirectly provided their over whelming support during this project work and the development of this paper.

REFRENCES

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, ''A View of Cloud Computing,'' Commun. ACM, vol. 53, no. 4, pp. 49-58.
[2] H. Biggar, ''Experiencing Data De-Duplication: Improving Efficiency and Reducing Capacity Requirements,'' Enterprise Strategy Grp., Milford, MA, USA, White Paper.
[3] C. Liu, Y. Lu, C. Shi, G. Lu, D. Du, and D.-S. Wang, ''ADMAD: Application-Driven Metadata Aware De-Deduplication Archival Storage Systems,'' in Proc. 5th IEEE Int'l Workshop SNAPI I/Os, pp. 29-35.
[4] A. Katiyar and J. Weissman, ''ViDeDup: An Application-Aware Framework for Video De-Duplication,'' in Proc. 3rd USENIX Workshop Hot-Storage File Syst., pp. 31-35.
[5] Y. Tan, H. Jiang, D. Feng, L. Tian, Z. Yan, and G. Zhou, ''SAM: A Semantic-Aware Multi-Tiered Source De-Duplication FrameWork for Cloud Backup,'' in Proc. 39th ICPP, pp. 614-623.
[7] A. Muthitacharoen, B. Chen, and D. Mazie `res, ''A LowBandwidth Network File System,'' in Proc. 18th ACM SOSP, pp. 174-187.
[9] S. Kannan, A. Gavrilovska, and K. Schwan, ''Cloud4HomeV Enhancing Data Services with Home Clouds,'' in Proc. 31st ICDCS, pp. 539-548.
[10] Maximizing Data Efficiency: Benefits of Global DeduplicationNEC, Irving, TX, USA, NEC White Paper.

