Two-way File Synchronization Mechanism in Cloud Storage System

Young Jun Yoo[1], Chang Gun Song[2], Seon Woo Lee[3], Jin Kim[1], Young Woong Ko[1]

[1.] Department of Computer Engineering, Hallym University, Chuncheon , Korea
[2.] Department of Ubiquitous Computing, Hallym University, Chuncheon , Korea
[3.] Department of Electronic Engineering, Hallym University, Chuncheon , Korea
{willow72, cgsong, senu, jinkim, yuko}@hallym.ac.kr; yuko@hallym.ac.kr

Abstract: In cloud storage system, file synchronization schemes are widely used for efficient use of storage system. It reduces network traffic and processing time extremely. However, there exist several weak point in handling file synchronization for multiple clients. In this paper, to tackle this problem, we propose a bidirectional file synchronization scheme to efficiently deal with file copying between the client and the server. Experimental result shows that the proposed scheme can reduce data traffic efficiently using bidirectional file synchronization.

[Young Jun Yoo, Chang Gun Song, Seon Woo Lee, Jin Kim, Young Woong Ko. **Two-way File Synchronization Mechanism in Cloud Storage System.** *Life Sci J* 2014;11(7):726-729] (ISSN:1097-8135). http://www.lifesciencesite.com. 106

Keywords: file system; deduplication; file synchronization; two-way

1. Introduction

With the explosion of the lots of data types such as multimedia, plain text, web data and documents, the cloud storage system is widely used to handle the data types. In the cloud system, deduplication is one of the main solutions to prevent accretion of duplicate data and bring cost saving in data centers. Deduplication technologies can further reduce the required storage capacity and can be used into various file systems. Data deduplication is technique for effectively reducing the storage requirement of backup feasible. Furthermore Data deduplication is a way to reduce storage space by eliminating data to ensure that only single instance of data is stored in storage medium. Data deduplication technique has also drawn attraction as a means of dealing with large data and is regarded as an enabling technology. Furthermore chunking based deduplication is one of the most effective, identical regions of data with references to data already stored on disk. Typically content-defined chunking and Fixed-sized chunking are main deduplication schemes in among the chunking based data deduplication approaches.

Nowadays, there are many mobile cloud storage systems where users try to synchronize their files through storage systems so that they update their files on any device. To accomplish file synchronization, lots of file synchronization schemes are proposed[1,2,3]. Generally, when a file sync program try to synchronize a source file to a server, it first have to find a target file by using metadata information such as file name, file modification time and etc. When it identifies a file that has been updated or changed it calculates the exact binary data within that file and sends only the changed information. With this approach, each file transfer only sends the data required to update the target file. If a target file is found on a server then the source file and the target file is synchronized using file synchronization algorithm such as Rsync. Without file synchronization scheme, the entire file would be sent to the server, even if only small parts of the file have changed.

Current file synchronization scheme has drawback handling multi-client synchronization where multiple clients try to synchronize its data on cloud storage. For example, suppose that two clients (*foo* and *goo*) try to synchronize data files with the file server and *file_A* is modified to *file_A'* on the client *foo*. While file synchronization process, data deduplication scheme only sends non-duplicated parts between *file_A* and *file_A*'. Now, file server updates metadata of *file_A'*. When the client *goo* tries to synchronize data files to the file server, the client will update *file_A* to *file_A'*. During this process, non-duplicated data chunks should be transferred to the client. However, usually, entire *file_A'* will be transferred to the client.

In this paper, to tackle this problem, we propose a bidirectional file synchronization scheme to efficiently deal with file update between the client and the server. We propose a two-way file synchronization scheme to enhance the performance of cloud storage system. The proposed algorithm utilize the metadata information of a client by temporally storing on server side. When another client tries to synchronize files, the server sends the metadata information to the client. With this approach, multi-clients can avoid hash computation overhead by exploiting pre-computed metadata information.

The rest of this paper is organized as follows. In Section 2, we describe related works about file synchronization system. In Section 3, we explain the design principle of proposed system and implementation details. In Section 4, we show performance evaluation result of the proposed system and we conclude and discuss future research plan.

2. Related Works

There are three different deduplication schemes; source-based, inline and post processing approach. The source-based approach performs data deduplication in the client side and the client sends only non-duplicated files or blocks to deduplication server. The client divides a file into several blocks and calculates hash key for each block. In the chunking process, the block can be divided into fixed size chunk or variable size chunk depending on chunking approach. The list of hash keys is delivered to the server and the server checks duplicated blocks by comparing the hash key with hash keys in the server. The server makes a non-duplicated block list and sends it to the client. Finally, the client sends the non-duplicated data blocks to the server. Inline approach performs data deduplication on the server side. A client sends file stream to the server then the server process deduplication work on the fly by chunking the file stream into blocks. In inline approach, the server has enough CPU resource and memory capacity for processing data deduplication. Finally, in post processing, the system first stores file stream on a temporary storage and performs data deduplication work later. Accordingly, the server needs additional storage device and all the computation can be delayed until there is available resource. Inline and post processing approach usually consume the system resource of a server while minimizing the client resource because all the deduplication work is processed on the server side.

Data deduplication research is actively studied in various university and research institute. One of the well-known data deduplication result is Venti[3] that is a network storage system using fixed-length chunking approach. The key idea of Venti is to divide a file into fixed blocks and check duplicated blocks using 160-bit SHA1 hash key. Venti can reduce storage capacity by eliminating multiple duplicated blocks that have the identical data, so duplicate data is easily identified and the data block is stored only once. In variable-length chunking, each block size is partitioned by anchor value that divides a file into variable size chunk. One of the well-known variable-length chunking is a LBFS[2] that exploits similarities between files or versions of the same file. LBFS avoids sending data over the network when the same data found in the server file system. LBFS achieves up to two orders of magnitude reduction in network bandwidth. In our previous research result, data deduplication system uses file modification pattern. This approach can detect how file is modified and what types of deduplication is best for data deduplication. Therefore, the optimal data deduplication policy can be applied to a file.

3. System Overview

Figure 1 shows the proposed system architecture. In the figure, client A and B is the client nodes using cloud storage and the server is a cloud storage node supporting two-way synchronization scheme.

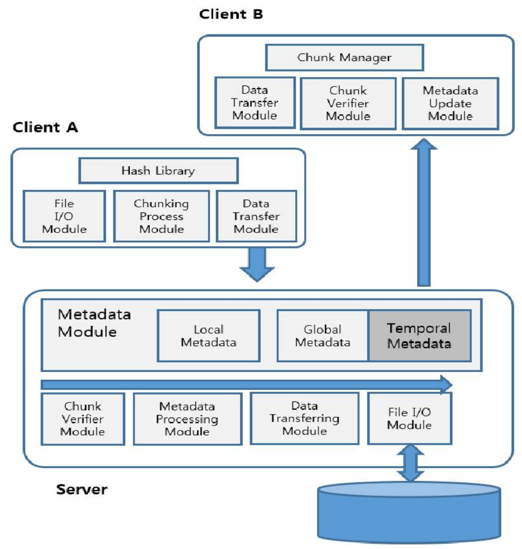

Fig. 1. Architecture of the proposed system

In this system, the client requests chunk hash lists to the file server and the server sends chunks list to the client. The client searches duplicated chunks by using variable-length chunking approach[4]. If the client finds duplicated chunks then it sends non-duplicated chunks list to the file server. By referencing this information, the file server can transfer non-duplicated data chunks to the client. In this processing steps, the file server saves metadata information (non-duplicated chunks list) temporarily. When there is another file synchronization requests, the file server can utilize the previous information without actual data deduplication processing.

In this work, we utilize the file similarity information[5,6] that has two tuples, hash key and file offset information. With that information, we can easily find duplicated region on a file by comparing hash key between two files. If there is same hash key, we use corresponding file offset where we apply fixed-length chunking, otherwise, we skip data deduplication. Therefore, the processing time of the proposed system is very short compared with variable-length chunking approach.

The key idea of this paper is applying file similarity information to find duplicated points between two files. In this work, we have to decide how much duplicated data blocks exist between two files. As a fast and efficient file comparison mechanism, we exploit the representative hash list that is used for evaluating the degree of similarity between two files. We made representative hash list for a given file by searching and composing the maximum hash list.

Figure 2 explains how similarity evaluation system works. First, a file stream is divided into variable chunks between anchors. The system calculates a hash key for fixed size blocks for each chunk. Conceptually, the sorting module lines up hash keys and makes a hash key list with ascending order or descending order. From the sorting list, we only take a few hash keys for representative hash values.

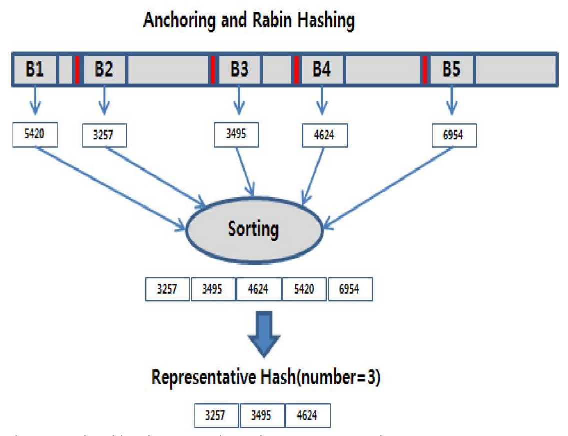

Fig. 2 Similarity evaluation processing step

As can be seen in figure 2, Rabin hash function is used for computing a hash key for a block. The Rabin hash starts at each byte in the first byte of a file and over the block size of bytes to its right. If the Rabin computation at the first byte is completed then we have to compute the Rabin hash at the second byte incrementally from the first hash value. Now that the hash value at the second byte is available then we use it to incrementally compute the hash value at the third, and continue this process. In this work, we have to sort the Rabin hash value and choose small number of maximum values as a representative hash. In this work, we made the representative hash list for all files before data deduplication. We extract one representative hash for 1 MByte therefore the amount of additional information for file similarity is not critical for metadata management.

To exploit file similarity scheme, we adapted delta compression for compress original file. If we find similar files on disk, we apply delta compression for each file with original files, therefore, we can minimize disk space for similar files.

4. Experiment Results

We have implemented the algorithm and conducted experiments to test the feasibility of the proposed algorithm. We have used a PC with 3.0GHz dual-core CPU and 4GB RAM, running Window 7. We made two test file sets using *lseek* function by patching data blocks. Set1 is composed of binary files such as executable files and multimedia files. Set2 is composed of documentation files such as Microsoft word, Powerpoint and Excel.

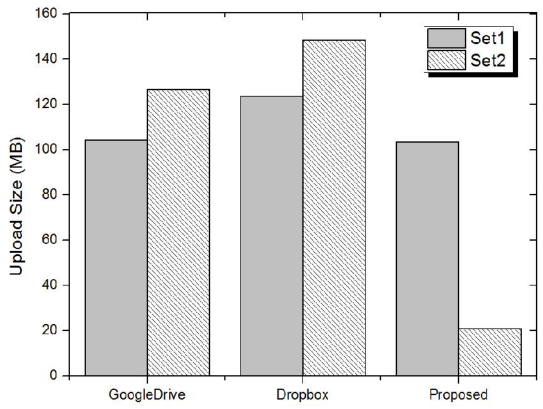

Fig. 3. Experiment result for uploading size

Figure 3 shows the uploading size result. As we can see this figure, traditional cloud storage system only supports one-way data deduplication therefore, total uploading size is very big. However, the proposed system can minimize uploading size up to 7 times for set2.

5. Conclusion

This paper proposes a two-way file synchronization scheme to enhance the performance of cloud storage system. The proposed algorithm utilize the metadata information of a client by temporally storing on server side. When another client tries to synchronize files, the server sends the metadata information to the client. With this approach, multi-clients can avoid hash computation overhead by exploiting pre-computed metadata information. We have implemented and tested the proposed algorithm. The results show that the proposed algorithm minimizes the overhead of file synchronization.

Acknowledgements:

This research was supported by Hallym University Research Fund, 2013(HRF-201311-009) and the MSIP, Korea, under the IT/SW Creative research program supervised by the NIPA(NIPA-2013-H0502-13-1063)

Corresponding Author:
Dr. Young Woong Ko
Department of Computer Engineering
Hallym University
Chuncheon, Gangwondo 200702, South Korea
E-mail: yuko@hallym.ac.kr

References

1. Aronovich, L. and Asher, R. and Bachmat, E. and Bitner, H. and Hirsch, M. and Klein, S.T., “The design of a similarity based deduplication system”, Proceedings of SYSTOR 2009, 2009.
2. Xia, W. and Jiang, H. and Feng, D. and Hua, Y., “Silo: a similarity-locality based near-exact deduplication scheme with low ram overhead and high throughput”, in ATC, 2011;1-14.
3. Quinlan, S. and Dorward, S., “Venti: a new approach to archival storage”, Proceedings of the FAST 2002 Conference on File and Storage Technologies, 2002.
4. Athicha M., Benjie C., and David M. “A Low-Bandwidth Network File System.” In Proceedings of the Symposium on Operating Systems Principles, 2001;174–187.
5. J. Kornblum, “Identifying Almost Identical Files Using Context Triggered Piecewise Hashing," Proc. 6th Ann. Digital Forensics Research Workshop Conf. (DFRWS 06), pp. 2006;S91–S97.
6. A. Andoni, P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in FOCS, IEEE Computer Society, 2006; 459–468.

5/26/2014