International Journal of Multimedia and Ubiquitous Engineering
Vol.9, No.1 (2014), pp.1-10
http://dx.doi.org/10.14257/ijmue.2014.9.1.01

Design and Implementation of Binary File Similarity Evaluation System

Sun-Jung Kim[2], Young Jun Yoo, Jungmin So[1], Jeong Gun Lee[1], Jin Kim[1] and Young Woong Ko[1]

[1]*Dept. of Computer Engineering, Hallym University, Chuncheon, Korea*
{willow72, jso, jeonggun.lee,jinkim, yuko}@hallym.ac.kr
[2]*Dept. of Ubiquitous Computing, Hallym University, Chuncheon, Korea*

sunkim@hallym.ac.kr

Abstract

In cloud storage system, when we search similar documentation files, keyword-based similarity evaluation scheme is well performed. However, if we want to find similar binary files then it is very difficult to satisfy user request. Because there is no widely used binary file search system that supports similarity evaluation among files. File similarity evaluation is essential for digital forensic and data deduplication field. In the file similarity processing time, the CPU consumption and resource overhead of memory are increased as the number of files increase. Moreover, as the file size is getting bigger, the overhead of metadata management is critical. In this paper, we suggest the similarity evaluation scheme using a hybrid chunking which reduce overall processing time of similarity evaluation. Experiment result shows that the proposed system can reduce processing time and data storage capacity effectively.

Keywords: *File, Synchronization, chunking, Hash, FLC*

1. Introduction

File similarity evaluation is used in various fields such as malicious file detection and data deduplication. In security area, malicious files are generated by modifying original executable file, so the file similarity search is used to identify file similarity or find malicious code from the executable programs within the system. Usually, the malicious file is very similar with original file because file patch is appended in the middle of original file. In data deduplication field, file similarity is very effective tools to eliminate duplicated data blocks. If we find similar files, then we can apply deduplication algorithm to the file. Data deduplication system handles similar files with various chunking approaches [1-3]. The well-known chunking approaches are fixed-length chunking and variable-length chunking (content-defined chunking). Deduplication approaches can save computing resources through detecting the file similarity and eliminating the duplicated region of a file. In file similarity searching system module [4, 5], the performance depends on hash comparison speed, therefore effective hash comparison is very critical.

In this paper, we propose file similarity evaluation system which determines similarity between client files and the server files. By searching high similarity files, we can reduce file synchronization overhead. This technique is much faster than traditional file synchronization systems which compare hash data of a client to the server one. The proposed scheme adapts

[1] *Corresponding author : yuko@hallym.ac.kr*

ISSN: 1975-0080 IJMUE

hybrid approach for data chunking, however it only hashes fixed-length data block by extracting hash values. Also the proposed scheme reduces hash comparison time by using representative hash scheme which generates a hash list that only contains key feature hash list of a file. One of the well-known file similarity evaluation schemes adapts representative hash approach. The key idea of representative hash is to extract set of hash keys from a file. In a concrete explanation, Rabin hash function [6] calculates hash key from a file and stores the hash key by shifting one byte step by step. Rabin hash function repeatedly generates hash key and inserts the hash key to the queue. The queue contains only several number of hash keys in ascending order or in descending order by configuration of the system. If Rabin hashing is finished, there remains several hash keys whose value is maximum or minimum. With this approach, representative hash scheme can predict how a file is similar with the other one. In representative hash scheme, when *A* file and *B* file have duplicated hash keys, this means that the files have duplicated region of data. However, to extract representative hash, all of the hash keys are compared with hash keys in the queue. This incurs high overhead on CPU and memory resources. To cope with this problem, in this work, we implemented file similarity evaluation system using hybrid approach that combines fixed-length chunking and variable-length chunking. The proposed approach chunks a file and finds anchor bytes. If it finds anchor bytes, it divides a file into fixed-sized chunk and calculates the hash key. The proposed system sorts this hash keys and assume as representative hash. Our approach can effectively produces the representative hash with small number of hash keys.

The rest of this paper is organized as follows. In Section 2, we describe related works about data deduplication system. In Section 3, we explain the design principle of proposed Byte-index Chunking 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. Background and 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. Design and Implementation of File Similarity Evaluation System

In this paper, we designed and implemented the file similarity evaluation scheme for cloud environment. As shown in Figure 1, the proposed system composed of server and client module. The client generates hash key by performing chunk process and sends the hash key list to the server. The server maintains metadata of each file using file similarity information. The role of file similarity evaluation system is to ensure that files in multiple locations have duplicated data blocks in certain rate. If a file is similar with a file on another location, then we can save network bandwidth by sending non-duplicated region of data. So, fast and accurate file similarity evaluation scheme is very important. In our approach, the purpose of file similarity evaluation is to keep minimum metadata for each file and provides high accuracy for predicting file similarity. The file similarity scheme can be widely used for backups or updating on storage systems and it is especially useful for mobile devices because if we reduce network packets then overall battery consumption will be minimized. To reduce data bandwidth, file similarity evaluation scheme has to find similar files in a rapid time when a client requests a file copying.

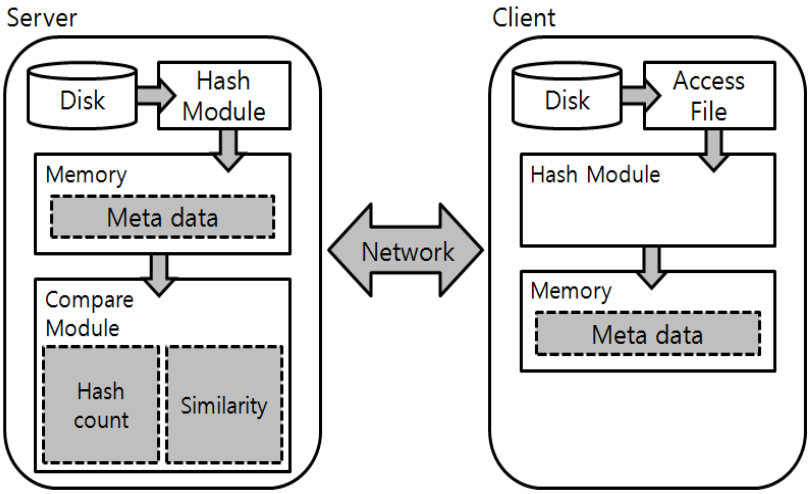


Figure 1. The proposed system architecture overview

In this paper, we adapt hybrid approach for data chunking. The proposed scheme only hashes fixed-length data block from anchor point. The anchor approach is very similar with variable-length chunking approach. However, in variable-length chunking, chunk block size is variable because it extracts a block between anchors. But, we only extract fixed size blocks from each anchor point. So, the proposed scheme can reserve the advantage of variable-length

chunking approach and reduce hash calculation time. Figure 2 shows the conceptual diagram of chunking method. Variable-length chunking divides blocks between anchors and the size of block is varying. However, the proposed scheme divides blocks between anchors, but the block has small size and fixed-length. The small and fixed size blocks takes few times during hash computation compared with variable-length approach.

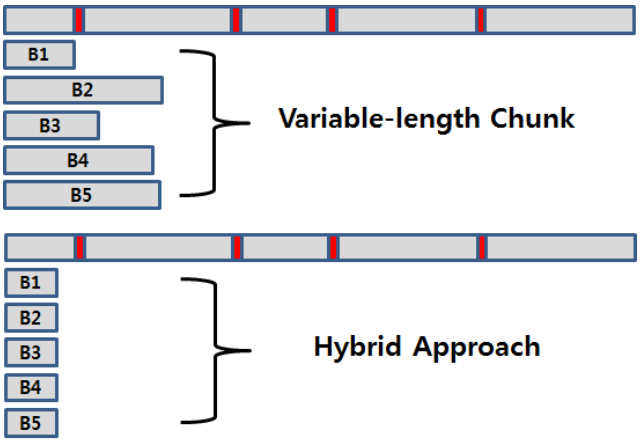


Figure 2. Comparison of chunking method

Figure 3 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.

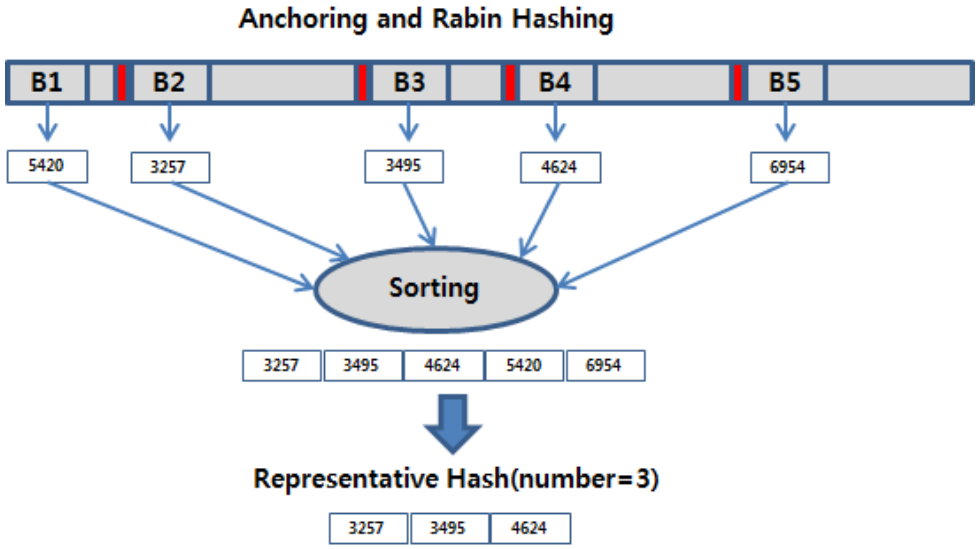


Figure 3. Similarity evaluation processing step

As can be seen in Figure 3, 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.

The proposed approach minimizes overall processing time for making similarity evaluation information. The main idea of this paper is to apply efficient hash comparison technique by reducing hash keys of file into small number of hash keys using representative hash selection. We designed and implemented the proposed file similarity evaluation scheme. Following *MetadataCompare* function is the key algorithm of the proposed system.

```
MetadataCompare ( sender )
foreach ChunkList serverchunk in serverChunkList  //file list of the server
  similarlist sim ← new similarlist()
  sim.filename ← serverchunk.filename
  for int i ← 0 i<serverchunk.Chunks.Count; i++ //hash list on the server
    foreach(Chunk chunk in clientdata.clientChunk)
    //hash list of the client
      if chunk.hash1 == serverchunk.Cunks[i].hash1 then
        clientdata.count++;
  sim.per ← clientdata.count / clientdata.clientChunk.Count() * 100
  clientdata.simlist.Add(sim)
clientdata.simlist.Sort(compare)
sender.SendData(simlist)
End
```

Figure 4. Pseudo code for similarity evaluation module on the server

The algorithm (Figure 4) shows how the proposed system evaluates similarity value between files. First, the algorithm reads file list of the server for computing similarity level. Second, for each file, it counts the same hash keys between two files and computes the similarity value. Finally, the similarity value list is transferred to the client.

Figure 5 shows the implementation result of the client of proposed system. A user selects a file from *browse* button and clicks "*the Send File VLC Data*" button. The selected file(file2.DAT) is transferred to the server and the server calculates file similarity value by comparing the file from the client and files in the server. The calculation result is returned to the client and presented on the left side of Figure 5. As we can see in Figure 5, the evaluation result is listed from a top similarity file to a low similarity file. In this result, file2.DAT is the same file between the client and the server. File1.DAT file has 79% of duplicated region compared with original file (file2.DAT in the client).

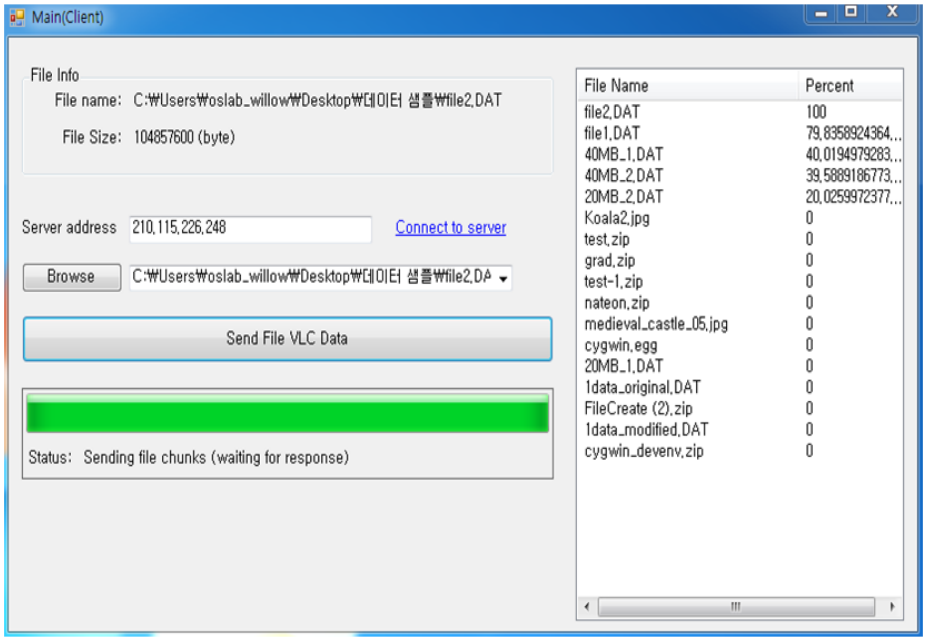


Figure 5. The implementation result of the client program

Figure 6 shows the implementation result of the server program. In this implementation, we stored several files in one folder and similarity evaluation is performed for these files. The user can select a file and do several operations such as delete, rename and move.

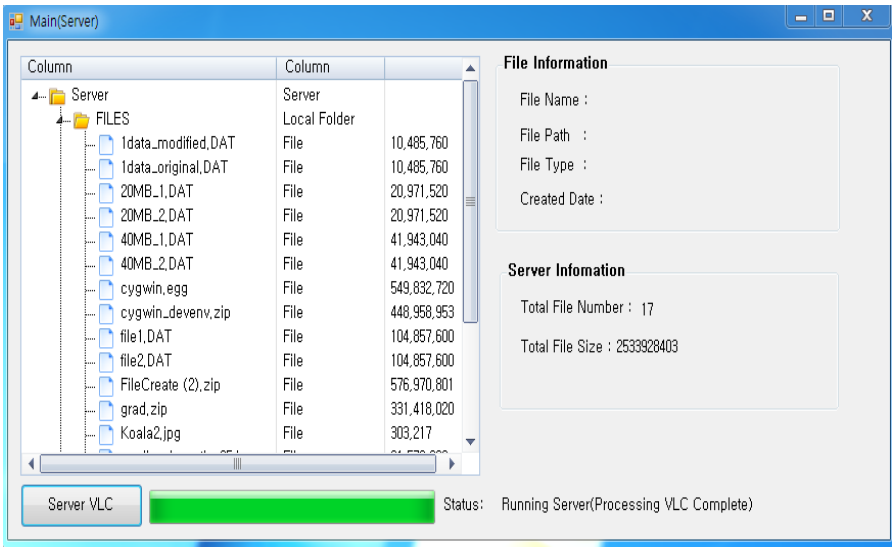


Figure 6. The implementation result of the client program

4. Performance evaluation

We now present experimental results that show the effectiveness of the proposed algorithm. To perform comprehensive analysis on the proposed algorithm, we implemented the client and the server on the platform that consist of 3GHz Pentium 4 Processor, WD-1600JS hard disk and 100Mbps network. We made experimental data set using patch that means data

block used for modifying a file in a random manner. In this experiment, we modified a file using *lseek* function in Linux system using random file offset and applied a patch to make test data file. For each run, we did multiple runs with different data sets, and plot the average resulting value. For algorithm comparison purpose, we also implemented variable-length chunking and fixed-length chunking. Similarity Evaluation Scheme(SES) is the proposed algorithm in this paper. In this experiment, original data is modified as 90%, 80% and 70% duplicates by Linux *dd* command. The experiment parameters of SES are same with VLC approach.

Table 1. File similarity experiment result

	FLC	SES
Number of hash	128000	479
Perform speed (sec)	0.305	0.003

Table 1 shows the performance result of execution speed. To see experiment result more detail, FLC file similarity approach produces 128000 hashes and SES approach only produces 500 representative hashes when performing the hash operation to the same file. The difference between performance speed of traditional FLC and SES are 0.3 sec and 0.003 sec, respectively.

Table 2. Performance result of error distribution

Actual file similarity	Similarity result (%)	Error (%)
90(%)	88	1.6
80(%)	81	1
70(%)	73	3.8

Table 2 shows errors of the proposed system for evaluation similarity result. From the experiment result, the proposed system can predict 88 % duplicates of data from actually 90% duplication data, and 1.6% loss 81% of 80 actual duplicates. Therefore, the proposed system is considered to be proved that shows very similar result to can determine file similarity in very high rate with fast time.

5. Conclusion

In this paper, we propose the file similarity evaluation scheme that determines similarity between files by using advanced representative approach. By searching high similarity files, we can reduce file synchronization overhead and storage capacity of the file server. The proposed scheme is much faster than traditional file synchronization systems which compare hash data of a client to the server one. The key idea is to adapt hybrid approach for data chunking that combines content-defined chunking and fixed-length chunking. Also the proposed scheme reduces hash comparison time by using representative hash scheme which generates a hash list that only contains key feature hash list of a file. Experiment result shows the proposed system can

Acknowledgements

This research was supported by Basic Science Research Program through the NRF funded by the MEST(.2012R1A1A2044694) , and this research was supported by Hallym University Research Fund, 2012(HRF-201209-024)

References

[1] K. Eshghi and H. Tang, "A framework for analyzing and improving content-based chunking algorithms", Hewlett-Packard Labs Technical Report TR, vol. 30, **(2005)**.
[2] A. Muthitacharoen, B. Chen and D. Mazieres, "A low-bandwidth network file system", ACM SIGOPS Operating Systems Review, vol. 35, no. 5, **(2001)**, pp. 174-187.
[3] S. Quinlan and S. Dorward, "Venti: a new approach to archival storage", In: Proceedings of the FAST 2002 Conference on File and Storage Technologies, **(2002)**.
[4] P. Kulkarni, F. Douglis, J. LaVoie and J. Tracey, "Redundancy elimination within large collections of files", In: Proceedings of the annual conference on USENIX Annual Technical Conference, USENIX Association, **(2004)**.
[5] H. M. Jung, S. Y. Park, J. G. Lee and Y. W. Ko, "Efficient Data Deduplication System Considering File Modification Pattern", International Journal of Security and Its Applications, vol. 6, no. 2, **(2012)**.
[6] M. O. Rabin, "Fingerprinting by random polynomials", Center for Research in Computing Technology, Harvard University, Report TR-15-81, **(1981)**.

Authors

Sun Jeong Kim

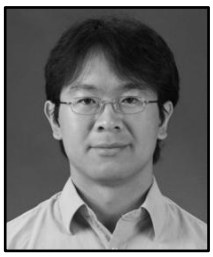
She received an MS degree in Computer Science at the Korea University in 1998, and a PhD degree from the Korea University in 2002. Since 2005 she has been working as an associate professor in Ubiquitous Computing at the Hallym University. Her research includes Scientific Visualization, Virtual Reality, and Bioinformatics.

Young Jun Yoo

He received the B.S. degree in computer engineering from Hallym University in 2011. He is currently pursuing his Master degree course in Department of Computer Engineering, Hallym University. His research interests include operating systems and file systems.

Jung Min So

He received the B.S. degree in computer engineering from Seoul National University in 2001, and Ph.D. degree in Computer Science from University of Illinois at Urbana-Champaign in 2006. He is currently an assistant professor in Department of Computer Engineering, Hallym University. His research interests include wireless networking and mobile computing.



Jeong Gun Lee

He received the B.S. degree in computer engineering from Hallym University in 1996, and M.S. and Ph.D. degree from Gwangju Institute of Science and Technology (GIST), Korea, in 1998 and 2005. He is currently an assistant professor in the Computer Engineering department at Hallym University.

Jin Kim

He received an MS degree in computer science from the college of Engineering at the Michigan State University in 1990, and in 1996 a PhD degree from the Michigan State University. Since then he has been working as a professor on computer engineering at the Hallym University. His research includes Bioinformatics and data mining.

Young Woong Ko

He received both a M.S. and Ph.D. in computer science from Korea University, Seoul, Korea, in 1999 and 2003, respectively. He is now a professor in Department of Computer engineering, Hallym University, Korea. His research interests include operating system, embedded system and multimedia system.

