Design and Implementation of Various File Deduplication Schemes on Storage Devices

Yong-Ting Wu, Min-Chieh Yu, Jenq-Shiou Leu
Department of Electronic and Computer Engineering
National Taiwan University of Science and Technology
Taipei, Taiwan
{M10302107, D10002103, jsleu}@mail.ntust.edu.tw

Eau-Chung Lee,
QNAP Inc., Taipei, Taiwan
ytlee@qnap.com

Tian Song
Electrical and Electronic Engineering,
Graduate School of Engineering,
Tokushima University, Tokushima City, Japan
tiansong@ee.tokushima-u.ac.jp

***Abstract*—As the smart devices revolutionize, people may generate a lot of data and store the data in the local or remote file system in their daily lives. Even though the novel computer hardware and network technologies can handle the demand of generating a big volume of data, effective file deduplication can save storage space in either the private computing environment or the public cloud system. In the paper, we aim at designing and implementing various file deduplication schemes on storage device, which are based on different duplication checking rules, including file name, file size, and file full/partial content hash value. Comprehensive experiment results show that a partial content hashing based file deduplication can have a better trade-off between the computation cost and deduplication accuracy.**

Keywords—file deduplication; cloud system; storage devices

I. INTRODUCTION

The emerging technical gadgets, like digital TV, smartphone, pad has rapidly driven a large volume of digit data. When the digital data are stored in a storage system, duplicated data may be conducted due to intended backups or unintended copies. By properly removing file redundancy in the storage system, the volume of information to manage is effectively reduced, significantly lessening the time and space required for file management. B. Hong, D. Plantenberg, D. D. Long, and M. Sivan-Zimet proposed their file deduplication scheme to improve the storage utilization of the storage area network [1]. D. R. Bobbarjung, S. Jagannathan, and C. Dubnicki then used the concept of file partitioning to increase the efficiency of the file deduplication scheme [2]. The aforementioned schemes are running with the online storage, which may not be suitable for the storage devices.

Besides, as the network applications have been widely developed and deployed in the world, application users would generate a lot of multimedia data in their daily lives, such as images or video clips captured by the digital cameras or cameras bundled in smartphones. Users then store them in the remote cloud system or the personal local storage. The demand for storage either in the local disk or in the remote storage farm hence increases. In addition, on account of the heterogeneity of the modern smart devices people may own, people more likely own duplicated multimedia data in many storage systems or even in the same storage system, resulting in an ineffective storage utilization and an inefficient search for some specific file in the system. Carrying out file deduplication schemes on the storage system can lessen the situation of wasting the space for duplicated files and increase the file search speed in the file system.

The most intuitive deduplication strategy is finding the files with the same file name or size. However, such a strategy may cause an inaccurate deduplicated result. Therefore, a hashing based file deduplication process is designed to increase the accuracy. However, a full content based hashing calculation may increase high computation cost [3]. A compromised way is taking a partial content based hashing calculation, which may bring a faster response to users, with a few sacrifices of deduplication inaccuracy [4, 5]. This work in the paper aims at how to design and implement the various file deduplication schemes for space saving. The detailed data structures, process flows for these schemes are also illustrated. Besides, a comprehensive evaluation results are depicted to validate the effectiveness of the implemented deduplication schemes.

The rest of the paper is organized as follows: Section II presents the data structures, process flows used in the three deduplication schemes. Section III details the experiment environment and the corresponding evaluation results. Finally, a brief conclusion is offered in Section IV.

II. DEDUPLICATION SCHEME IMPLEMENTATION

We briefly design three intuitive approaches to implement the file deduplication schemes on storage devices, including by the filename, by the size, and by the MD5 (Message-Digest algorithm number 5) hash value [6]. The introduction of the data structures and processing flows used is shown below.

A. *Data Structures*

To implement the file deduplication system, we need to define the data structures first, and then use the data structures to carry out the file deduplication procedure.

1) By the filename: This is the most intuitive and easiest approach of three deduplication schemes. The user may copy the file into another folder but forget to delete the old one. Hence, the main goal of this approach is to find out and show

QSHINE 2015, August 19-20, Taipei, Taiwan

DOI 10.4108/eai.19-8-2015.2260903

the properties of the files with the same filename. Then, the user can decide whether to delete the duplicated files or not.

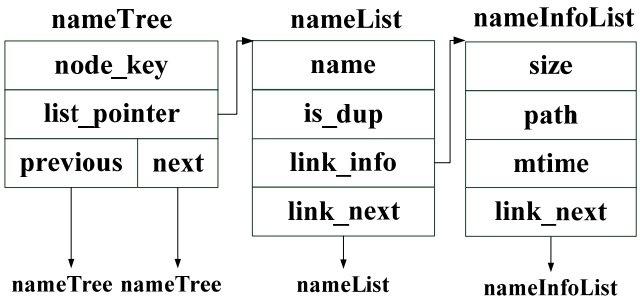

Fig. 1. The data structure in the filename based approach

Fig.1 shows the data structure of the approach. An node would be generated by the deduplication procedure and it contains the nameTree, nameList and nameInfoList which are shown in Fig.1.

a) nameTree: This node is the header in the filename based approach. The procedure would convert the filename into ASCII (American Standard Code for Information Interchange) values and store the summation value in *node_key*. The address of nameList is stored in *list_pointer*. Moreover, The addresses of previous and next nameTrees are stored in *previous* and *next* respectively.

b) nameList: The list is used to store the filename (*name*) of files with the same *node_key*. The deduplication scheme would change the value of *is_dup* to note if the filename is duplicated. The address of nameInfoList is stored in *link_info*. Furthermore, the address of next nameList is stored in *link_next*.

c) nameInfoList: The main property of the file is stored in this list, including the filesize(*size*), the filepath(*path*), and the file last modified time(*mtime*). Additionally, the address of next nameInfoList is stored in *link_next*.

2) By file size: The approach is based on an intutive idea that the same file has the same filesize. The user may copy the file into another location and change its name, but forget to delete the original one. Therefore, the approach would find out the files with the same filesize, showing the details on the user interface so that users can decide which duplicated file needs to be deleted.

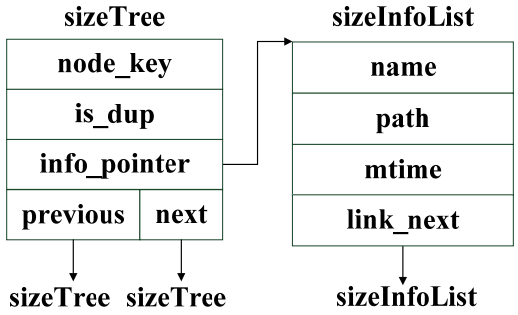

Fig. 2. The data structure in the file size based approach

Fig.2 shows the data structure of the approach. An node created in the approach would contain the sizeTree, and sizeInfoList which are shown in Fig.2.

a) sizeTree: This node is the header in the file size based approach. The procedure would store the file size value in *node_key*. The deduplication scheme would change the value of *is_dup* to note if the file size is the same. The address of sizeInfoList is stored in *info_pointer*. Moreover, The addresses of previous and next sizeTree are stored in *previous* and *next* respectively.

b) sizeInfoList: The main property of the file is stored in this list, including the filename(*name*), the filepath(*path*), and the file last modified time(*mtime*). Additionally, the address of next sizeInfoList is stored in *link_next*.

3) By the MD5 hash value: In order to avoid deleting the indepedent files with the same file size, the scheme is designed based on the calculated hash value. The scheme would calculate the MD5 hash value of first and last 10 percent of the file content, or even the compelte file content to improve the accuracy of duplication check.

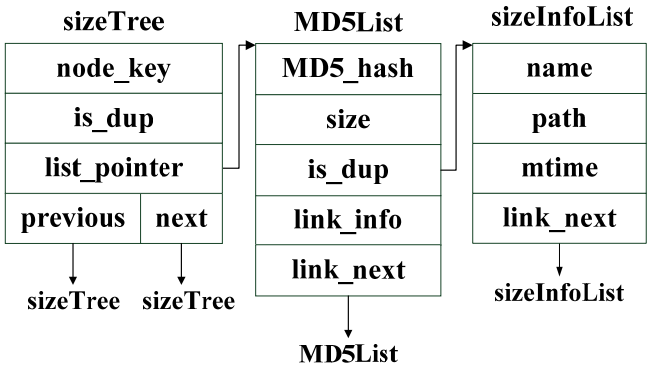

Fig. 3. The data structure in the MD5 hash value based approach

Fig.3 shows the data structure of the approach. Since the approach is an extended approach of the file size approach, a node created in the approach would contain the sizeTree MD5List, and sizeInfoList which are shown in Fig.3.

a) sizeTree: The address of MD5List is stored in *list_pointer*. The rest parameters are the same as the data structures in the file size based approach.

b) MD5List: The list is used to store the calculated MD5 hash value (*MD5_hash*) of files with the same *node_key*. The deduplication scheme would change the value of *is_dup* to note if the MD5 hash value is duplicated. The address of sizeInfoList is stored in *link_info*. Moreover, The address of next MD5List is stored in *link_next*.

c) sizeInfoList: The list is identical with *sizeInfoList* which is mentioed in the file size based approach.

B. *Processing Flows*

The processing flows of all designed approaches are shown in Fig. 4, 5, and 6. Since the main objects of three deduplication schemes are the same, the processing flows are similar.

1) *Find the duplicated filename among files:* For the filename based approach, the process would choose one file first in the selected directory and use the ASCII code to convert the filename of the chosen file as *node_key*. Then, the process would search all nameTree to find out the existing nameTree with the calculated *node_key*. Once the nameTree with same

node_key is found, the process would check if the filename in the nameList is the same as the chosen file. If yes, the process would insert the new nameInfoList after that one in the existing nameTree, and change the value of *is_dup* to note that the duplicated file with the same filename is found. If no, the process would take the chosen file as a new file, insert the new nameList next to the old one, and then store the property of the new file into its own nameInfoList.

However, if there is no any existing nameTree with the calculated *node_key*, the process would create a new instance of nameTree and store the file information into nameList and nameInfoList. Subsequently, the process would continue until there is no any unchecked file in the selected directory.

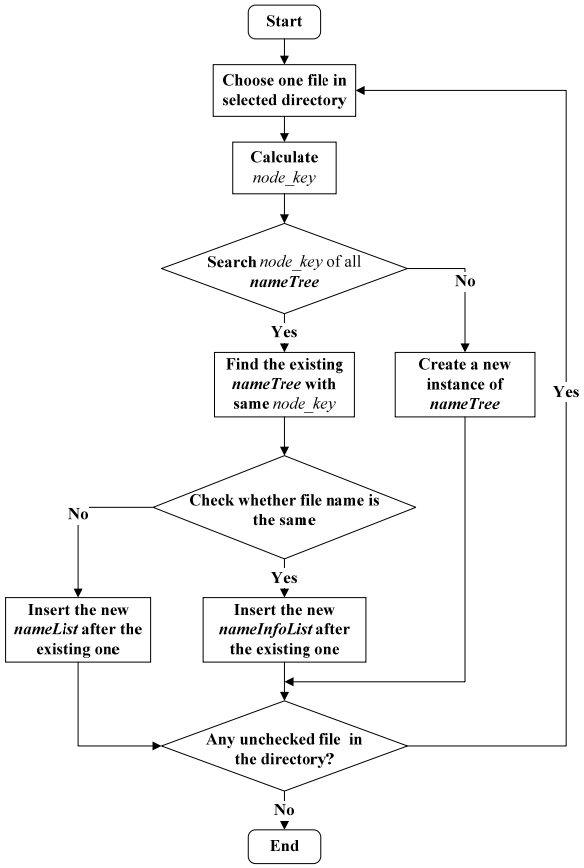

Fig. 4. The processing flow in the filename based approach

2) *Find the same file size among files:* For the file size based approach, the process would choose one file first in the selected directory and store the file size value of the chosen file as *node_key*. Then, the process would search all sizeTrees to find out if there exists one sizeTree with the same *node_key*. Once the sizeTree with the same *node_key* is found, the process would insert the new sizeInfoList after that one in the existing sizeTree, and change the value of *is_dup* to note that the duplicated file with the same file size is found.

However, if there is no any existing sizeTree with the calculated *node_key*, the process would create a new instance of sizeTree and store the file information into nameList and nameInfoList. Subsequently, the process would loop until there is no any unchecked file in the selected directory.

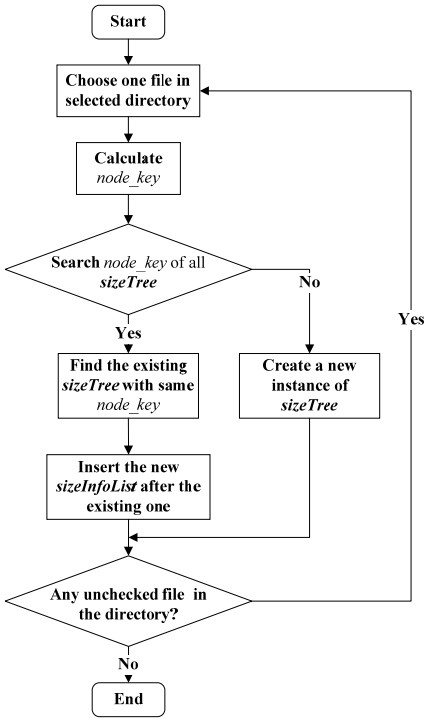

Fig. 5. The processing flow in the file size based approach

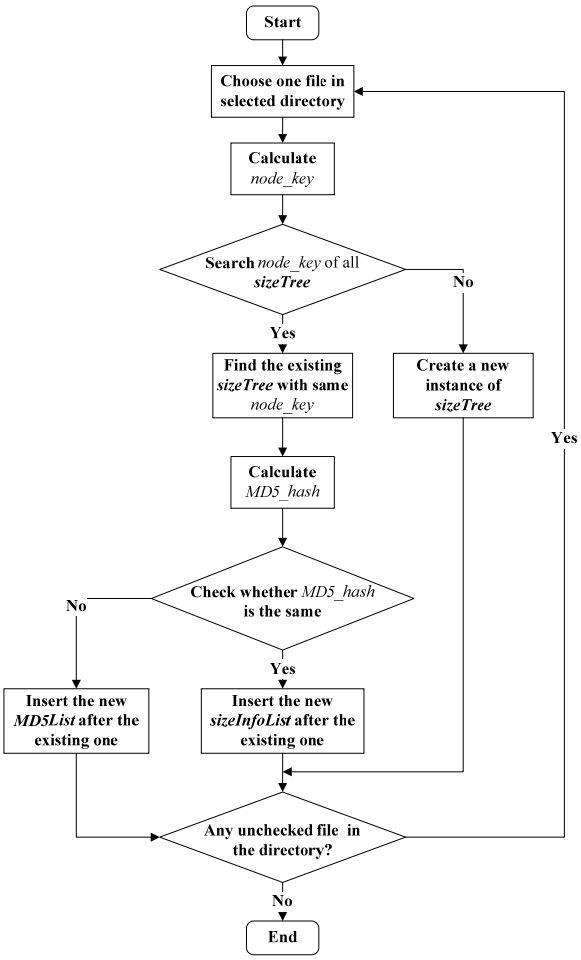

Fig. 6. The processing flow in the MD5 hash value based approach

3) *Find the same MD5 hash value among files:* Since the approach is an extension of file size based approach, the most part of the MD5 hash based approach is identical to the file size based one. Once the sizeTree with same *node_key* is found, the process would calculate *MD5_hash* of the chosen file. After that, the process would check if *MD5_hash* in the MD5List is the same as the one of the chosen file. If yes, the process would insert the new sizeInfoList after that one in the existing sizeTree, and change the value of *is_dup* to note that the duplicated file with the same MD5 hash value is found. If no, the process would take the chosen file as a new file, insert the new MD5List next to the old one, and then store the file property of the new file into its own sizeInfoList.

4) Time Complexity of file deduplication algorithms: Assuming that the storage devices have N files, the time complextiy of the filename based approach would be O(N log N), since the algorithm would maintain a tree structure when checking each file, and the time complexity of manipulating a tree structure is O(log N). Meanwhile, because of a similar process flow, the complexity of the file size based approach would also be O(N log N).

For the MD5 hash based approach, the time complexity can be divided into two parts: the complexity of the first part is O(N log N) since the MD5 hash based approach is an extension of file size based one, and the complexity of the second part is O(n) due to the MD5 hash value calculation process. Hence, The complexity of the MD5 hash based approach would be O(N log N) + O(n). In addition, the worst-case complexity of the second part may be $\frac{(1+N)N}{2} = O(N^2)$, since the MD5 hash values of all files need to be calculated.

III. Evaluation Results

A. *Evaluation Environment*

To evaluate the performance of implemented schemes, we used a typical network-attached storage (NAS) device as the storage device to run the these three deduplication check schemes. The NAS used in the evaluation is QNAP NAS TS-269L, and its specification is shown in Table I.

TABLE I. EVALUATION ENVIRONMET SPECIFICATION

Unit	Detail
CPU	Intel® Atom™ 1.86 GHz Dual-core Processor
HDD	TOSHIBA DT01ACA300, SATA III, 7200rpm, 3TB
Memory	1GB DDR3-1066 RAM

In the evaluation, we design three different testing scenarios to test the three file deduplication schemes. The testing scenarios are distinguished by the file size and named by KB, MB, and GB. The diagram of testing scenario is shown in Fig.7.

For the KB scenario, the total file size of the first sub-folder 1K is 1 kilo-byte, the second sub-folder 10K is 10 kilo-bytes. Respectively, the total file size of each under the i sub-folders is $S_{kb} = 10i$, and the values of i are ranged from 2 to 99. There are 15 group-folders in each sub-folder. Moreover, each group-folder has 5 end-folders. In each end-folder, there are 1 group-wide duplicated file, 9 different files, and 1 fixed duplicated filename file. For an example of a group-wide duplication file, each end-folder which is the member of group-folder 1 under the sub-folder 1K has one group-wide duplication file (dup_file 1KG1). The total file amount of the scenario is 82,500.

For the MB scenario, the total file size of the first sub-folder 1M is 1 mega-byte, and the second sub-folder 10M is 10 mega-bytes. The total file size of each under the j sub-folders is $S_{mb} = 50j$ mega-bytes, and the value of j is ranged from 1 to 19. There are 5 group-folders in each sub-folder. Additionally, each group-folder has 5 end-folders. In each end-folder, there is 1 group-wide duplicated file, 9 different files, and 1 fixed duplicated filename file. The scenario contains 5775 files totally.

For the GB scenario, the total file size of each sub-folder n is $S_{gb} = n$ gigabytes, and the value of n is ranged from 1 to 5. There are only 2 group-folders in each sub-folder. Additionally, each group-folder has 2 end-folders. In each end-folder, there are a group-wide duplication file, 4 different files, and a duplicated filename file. The total file amount in the scenario is 120.

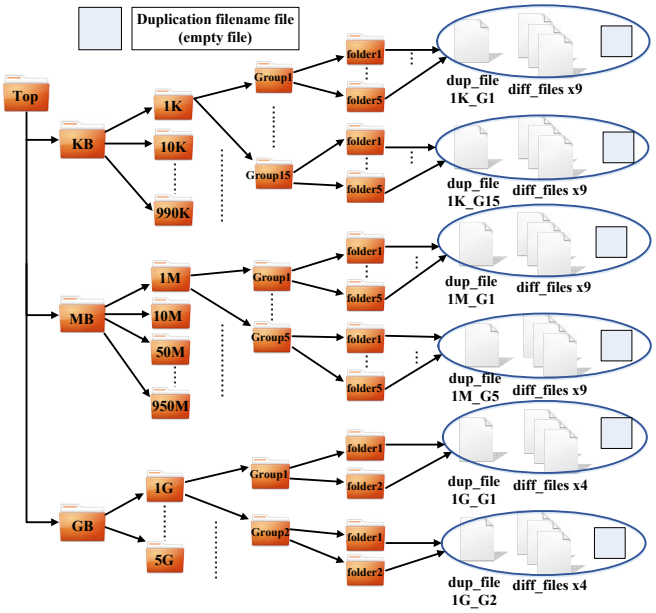

Fig. 7. The diagram of the testing scenario

B. *Evaluation Results*

The evaluation results conducted by the three file deduplication schemes are shown in Table III. In table III, "Filename" represents the filename based duplication approach; "File Size" represents the file size based duplication approach; "MD5" represents the MD5 hash based approach; MD5(Partial) represents the scheme only using a partial of the file content, which is listed in table II, to calculate the MD5 hash value, while MD5(Full) uses the complete file content to calculate the MD5 hash value.

TABLE II. PARTIAL HASHING SPECIFICATION

File size	The percentage of file content used for hash value computing
< 1MB	100%
≥ 1MB & < 1 GB	1%
≥ 1GB	0.1%

TABLE III. EVALUATION RESULT

	Filename	File Size	MD5 (Partial)	MD5 (Full)
Time usage	1m19s	1m21s	5m37s	58m17s
CPU usage ratio	8.2%	12.2%	24.9%	24.9%
Memory usage ratio	6.1% (60MB)	6.3% (62MB)	8.6% (85MB)	8.6% (85MB)

In the table III, we can observe that the filename based approach and the file size based approach perform faster compared to the MD5 hash value based one, as the time complexity of MD5 hash value based approach is greater than other two ones. Besides, all evaluation results of the filename and the file size based approaches are similar due to the complexities of the filename based one and the file size based one are identical. Further, the MD5 hash value based approaches has the highest CPU utilization and memory usage ratio.

IV. CONCLUSION

An effective file deduplication scheme can facilitate to save space on the storage devices. We have successfully designed and implemented three file deduplication schemes on the storage device in this study. The system structures and processing flows of three approaches are shown. Finally, we use an instance of storage device to evaluate the three file deduplication schemes, and the evaluation results have been discussed.

Acknowledgement

The authors gratefully acknowledge the financial support from the "Aiming For the Top University Program" funded by Ministry of Education, Taiwan.

REFERENCES

[1] B. Hong, D. Plantenberg, D. D. Long, and M. Sivan-Zimet, “Duplicate Data Elimination in a SAN File System,” In 21st *IEEE Conference on Mass Storage Systems and Technologies (MSST)*, pp. 301-314, Apr. 2004.

[2] D. R. Bobbarjung, S. Jagannathan, and C. Dubnicki, “Improving duplicate elimination in storage systems,” *ACM Transactions on Storage (TOS)*, vol 2(4), pp. 424-448, 2006.

[3] D. Meister, and A. Brinkmann, "Multi-level comparison of data deduplication in a backup scenario," in *Proceedings of 2nd The Israeli Experimental Systems Conference (SYSTOR'09)*, pp. 1-12, May 2009.

[4] V. Henson, "An analysis of compare-by-hash," in *Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS-IX)*, 2003, pp. 13-18.

[5] J. Malhotra, and J. Bakal, "A survey and comparative study of data deduplication techniques," in *Pervasive Computing (ICPC)*, 2015 International Conference on , pp. 1-5, Jan. 2015

[6] R. Rivest, RFC 1321: The MD5 Message-Digest Algorithm, Network Working Group, Apr. 1992.