Mobile Netw Appl (2017) 22:40–50
DOI 10.1007/s11036-016-0677-9

Design and Implementation of Various File Deduplication Schemes on Storage Devices

Kuan-Wu Su[1] · **Jenq-Shiou Leu**[1] · **Min-Chieh Yu**[1] · **Yong-Ting Wu**[1] · **Eau-Chung Lee**[2] · **Tian Song**[3]

Published online: 15 January 2016

Abstract As smart devices are revolutionized in recent years, people may generate enormous amount of various sized data and store them in the local or remote file system in their daily lives. With cheaper and easy to use private cloud storage appliances helping to handle the increasing demand of storing and sharing big volume of data, effective file deduplication schemes can greatly increase the space efficiency in private cloud storage systems as well as preserve network bandwidth. In the paper, we aim at designing and implementing several file deduplication schemes built in the private cloud storage appliance, based on different duplication checking rules, including file name, file size, and file partial/full content hash value. Experiment results show using partial content hashing based file deduplication scheme achieves a reasonably balanced performance without overutilized limited local computational resources.

Keywords File deduplication · Cloud system · Storage devices

✉ Jenq-Shiou Leu
jsleu@mail.ntust.edu.tw

Eau-Chung Lee
ytlee@qnap.com

Tian Song
tiansong@ee.tokushima-u.ac.jp

[1] Department of Electronic and Computer Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan

[2] QNAP Inc, Taipei, Taiwan

[3] Department of Electrical and Electronic Engineering, School of Engineering, Tokushima University, Tokushima, Japan

1 Introduction

The emerging technical gadgets, like digital TV, smartphone, pad have rapidly driven a large volume of digit data demand for sharing, exchanging, and data storage. Many applications have been widely deployed in them and generate a lot of multimedia data in people's daily lives, such as images or video clips captured by the digital cameras or cameras bundled in smartphones, driving the demand for more remote cloud storage. In addition, on account of the heterogeneity of the modern smart devices, people are more likely to own duplicated multimedia data in much different storage systems, resulting in ineffective storage utilization.

The increase of demand for remote storage give popularity to many public cloud storage services but most of them with fairly limited storage space and cumbersome interface, hence many turn to private cloud storage appliances equipped with extra computational power that can run private cloud applications and services. However, the large amount of files from mixed sources are stored and shared by various storage services as active data, thus duplication occurs when different applications and services synchronize and sometimes generate different copies in the local private cloud storage file system, most of the time spread in different folders under deep directory sub-folders. Meanwhile, multiple clients using the same private cloud storage may unintentionally create additional backups and copies as well.

While deduplication appliances have been widely adopted by enterprises and offices in their own storage area network (SAN) [1], network attached storage (NAS) appliances are just become popular for smaller private clouds, such as, in a home setup environment using just a single NAS device of limited computational resource and limited network bandwidth to alleviate all incoming requests. With proper deduplication scheme running, the volume of information to manage is

effectively reduced, and significantly lessening the time and space required for file management. Hong, et al. [2] proposed their file deduplication scheme to improve the storage utilization of SAN. Bobbarjung, et al. then used the concept of file partitioning to increase the efficiency of the file deduplication scheme [3]. The aforementioned schemes are running with the online storage, which may not be suitable for small private cloud storage. Min, et al. [4] proposed a deduplication scheme divided files into chucks and manage the chuck SHA-1 hash value as fingerprints based on file content types. More recently, Li, et al. [5] presented several new deduplication constructions in a hybrid cloud architecture. Stanek, et al. [6] focus on security in file deduplication, and showed an encryption scheme based on file popularity in cloud storage environment.

However, a lot of traditional deduplication schemes focus on scenarios where files are in cold storage state and are collected via a large body of users [3], or on the other hand dealing with inline active files where latency and backup speed is the primary concern [4]. Rarely do they consider a small group of cooperated users sharing limited private cloud storage resources, and these files are neither totally inactive nor always active every day, but most of the files are active within a year, while only about 20% are active within a month [7]. At the same time, the emerging technical gadgets enable just a handful of users to generate enough multimedia data that would fill up a shared private cloud media quickly without deduplication schemes build-in.

Although most recent studies focused more on cloud storage deduplication across many users, and naturally issues like file security, illegal access, violation of access, and identifying unauthorized files via side-channels [8], etc. But for a private cloud with limited clients, a reasonable way to share file is to dump shared files in a shared file pool, where everyone in the group has access. In the meantime, the tragedy of the digital commons arises where no one wants to manage the space quota of this limited shared storage space, while a lot of the shared files are duplicated over and over by different users before been accessed again months, even years later, and by then no one is sure these files are duplicated. Implementing an inline deduplication scheme during access is a way to solve this issue, but a lot of moderate NAS appliances do not have such feature, while a lot of offline deduplication scheme is too costly to run as constant background processes.

In this paper, we present the implementation and evaluation of a file deduplication scheme that can run reasonably using only few computational resources in such mixed cloud storage scenario, where deduplication is not a constant need for inline file access, but could be run manually on demand as applications or automatically in the background for private cloud storage appliances such as NAS devices.

The rest of the paper is organized as follows: Section 2 provides a more detail discussion about the issues regarding deduplication in mixed cloud systems environment. Section 3 presents the data structures, process flows used in our deduplication schemes, followed by theoretical analysis in Section 4. In Section 5, the experiment environment with corresponding evaluation results are presented, and finally, a brief conclusion in Section 6.

2 Deduplication issues

Below is a representation of the scenario described in Section 1 as Fig. 1. As described above in introduction, in a hybrid/mixed cloud environment of limited group of users, they can share a common private cloud file pool locally, while at the same time different users may use different public cloud services. For some files are uploaded directly to the shared private cloud, but other files mostly from roaming devices are updated to public cloud services, and then automatically synchronized to the private cloud server, such as a NAS appliance. Even so, some NAS appliances have their own API or applications for mobile devices to directly synchronize file to private cloud, where some users may choose to synchronize their files via USB or other means to local PCs before these files are uploaded to the private cloud. All these different synchronization and upload behaviors from users often create many different copies of the same file in the shared file pool.

There are many storage deduplication schemes and designs can be implemented to reduce storage cost [9]. Most of them have trade-off between utilization and performance as shown in Fig. 2.

In order to implement a scheme under the mixed cloud storage scenario in Fig. 1, we need to find a balance between utilization of storage space in limited private cloud storage, at the same time considering the computational resource required. Emphasize too much on inline primary storage would cost too much computational resources where a modest private cloud storage appliances may not have, but too much on unpopular files and treat it like traditional inactive cold storage, it would likely needs complex multi-stages file system structure, or unnecessary access control via constant transmission bandwidth to check redundancy.

More than often the issue is not to design a complex or complete top-down system, but simply a flexible and adaptable application that can be run concurrently with other services within the private cloud storage appliance. At the same time without needing to redesign or re-implemented existing file access scheme.

There are several important deduplication considerations need to be addressed:

– The first consideration is the data granularity, how large are the files or chunks of files needed to be deduplicated and what's their file size distribution, also whether to chop files into equal sized blocks to increase

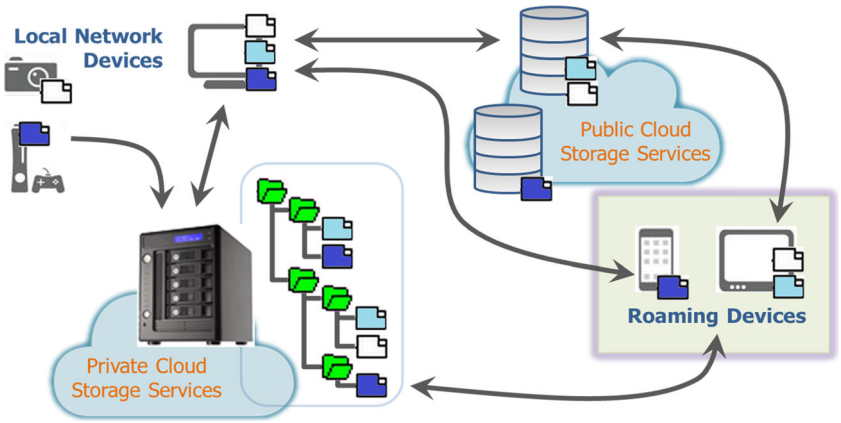

Fig. 1 Mixed cloud storage scenario

deduplucation efficiency. With more and more data generated by multimedia enabled devices, and increasing resolution in displays, the overall size of the files needed to be deduplucated now ranging from KB, to MB, all the way to GB level files are common with larger files raise in overall ratio. Even with the fast storage medium and high speed internet bandwidth, a very large file or a boundle of files needed to be synchronized might take minutes even hours to be transfered between different storages.

– The second consideration is storage type. From active data stored in primary storage system like local hard drive to memory card, or SSD in active working devices or computer systems, to secondary storage media like NAS devices, or private cloud storage appliances, to off-site like cold storage data farms or removable permanent storage mediums. For the most part, active files are most often stored in primary storage system, where they are constantly modified, rewritten, or generated, while data in cold storage is mostly immutable. But in the middle, the data in secondary storage is semi-active. They are constantly synchronized with active data, or accessed via web APIs, and become achieved from time to time.
– The other consideration is when deduplication scheme is performed. Inline or in-band deduplication means the checking of duplicated files or blocks of data is performed before written into storage medium. While offline or off-band, sometimes referred as post processing scheme performed deduplication after they are written into the storage medium. However offline deduplication temporarily requires more storage space.
– And finally the consideration of indexing, giving each file/block fingerprints, can greatly increase the speed of deduplication checks, and let inline deduplication have higher spatial utilization, or less bandwidth during data transfer between different level of storages. But for large quantities of data, this step needs to be achieved in a limited time window, and shouldn't consume too much computational resources. Another benefit of indexing is to make sure only unique data is sent to cloud storage services, also reduce the unnecessary comparison of the whole files after them reach the cloud storage servers. The use of indexing can also help increase the efficiency of file searching across multiple cloud storages.

In this paper, we choose the scenario as shown in Fig. 1 where a secondary storage medium is used in a private cloud,

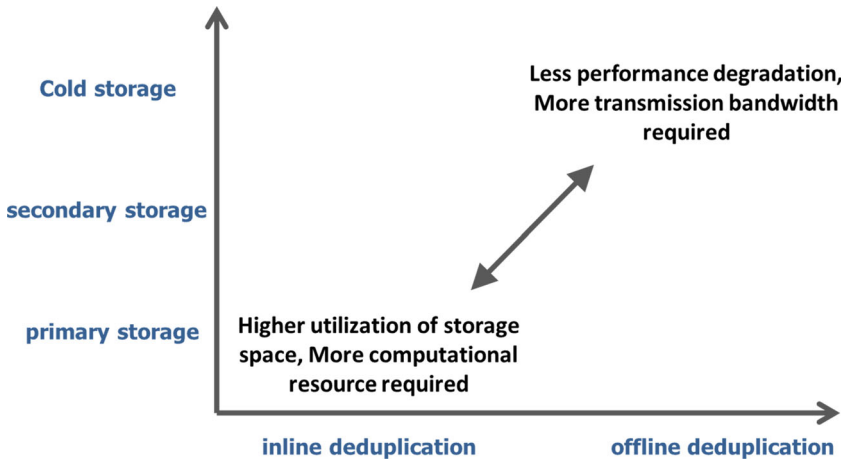

Fig. 2 Example of deduplication trade-off

where multiple users can access and share data under the same private cloud with network attached storage (NAS), where their smart devices can roam outside of local area network. But these devices may also be linked to other public cloud storage services as their default cloud storage option with limited online storage space, hence the secondary private cloud storage space becomes the backend services where less active and large files are shared and stored. Due to the nature of different users might prefer different public cloud storage services, the private cloud storage has to account for the uncoordinated and sometimes unavoidable user activities that generate redundancy files by accident.

Since the NAS device has its own processing power and enough storage space usually up to several TB of storage space, but the computational resources have to be shared by various applications and services running in the background, hence an offline post processing scheme is preferred and often times a trade-off between accuracy and efficiency needs to be struck.

The most intuitive deduplication strategy is finding the files with the same file name or size. However, such a strategy may cause an inaccurate deduplication result. Therefore, an index approach uses hash function in deduplication post process should be included. However, a full content based indexing with complex hash calculation may consume too much computational resources [10]. A compromised way is by taking a partial content based hashing calculation, which should result in faster response time, with some sacrifices [11, 12]. In this paper, we implement 4 types of post processing deduplication schemes running as a client enabled application with a Web API, and perform comprehensive evaluation to depict their effectiveness.

3 Deduplication scheme implementation

The detailed data structures and process flows for four post processing deduplication schemes on private cloud NAS device, including *by the filename*, *by the size*, *by partial and full file hash values* are illustrated as follows.

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 deduplication schemes. The user may copy files to another folder but forget to delete the old ones. Hence, the main goal of this approach is to find and show the properties of those files with the same filename

Figure 3 shows the data structure of the filename based approach. An node is generated by the deduplication procedure and it contains the *nameTree*, *nameList* and *nameInfoList*. The definitionsof these data structures are listed below.

a) *nameTree*

This node is the header in the filename based approach. The procedure converts 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 file size, and when large files are common, the chances of different files as the same size is low. The user may copy files to another location and change their names, but forget to delete the original ones. Therefore, the approach is able to find out those files with the same file size, and showing the details in the user interface.

Figure 4 shows the data structure of the approach. An node created in the approach contains the *sizeTree*, and *sizeInfoList*. Their definitions and functions are listed below.

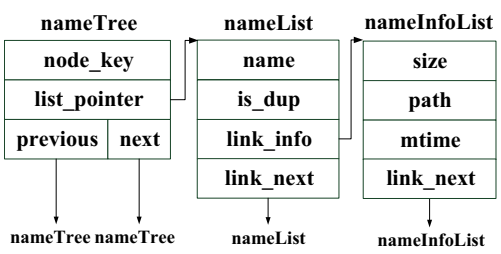

Fig. 3 The data structure in the filename based approach

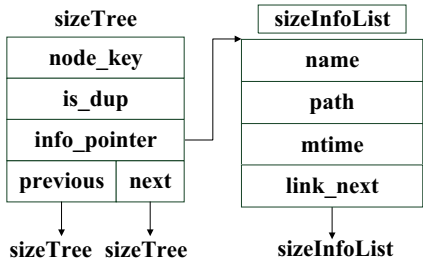

Fig. 4 The data structure in the file size based approach

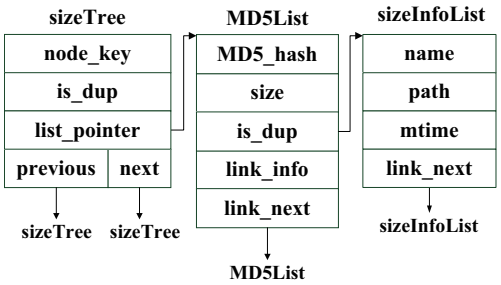

Fig. 5 The data structure in the MD5 hash value based approach

a) *sizeTree*

This node is the header in the file size based approach. The procedure stores the file size value in *node_key*. The deduplication scheme changes 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 hash values

In order to avoid deleting different files of the same file size as a false positive match, the calculated hash values are used to determine wither for those files with the same file size are indeed different files. The scheme calculates the MD5 (Message-Digest algorithm number 5) hash values [13] of the file content, with both the *representative block* as part of the file, or the compelte file hash values as indexes to improve the accuracy of the duplication check. The size of the representative block depends on the file size. If the file size is smaller than 10KB, the block size equals to the file size, if larger than 10K, the size of the representative block is 10KB.

Figure 5 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 contains the *sizeTree*, *MD5List*, and *sizeInfoList*, which are listed below.

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* mentioed in the file size based approach.

B Processing Flows

The processing flows and descriptions of all designed approaches are as follows:

1) Find the duplicated filename among files

For the filename based approach, the process chooses 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 searches 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 there is a match, the process inserts the new *nameInfoList* after the one in the existing *nameTree*, and changes the value of *is_dup* to note that the duplicated file with the same filename is found. If no match founds, the process takes the chosen file as a new file, insert the new *nameList* next to the old one, and then stores 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 creates a new instance of *nameTree* and stores the file information into *nameList* and *nameInfoList*. Subsequently, the process would continue until there is no any unchecked file in the selected directory. The entire processing flow is shown in Fig. 6.

2) Find the same file size among files

For the file size based approach, the process chooses 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 *sizeTree* to find out if there exists one *sizeTree* with the same

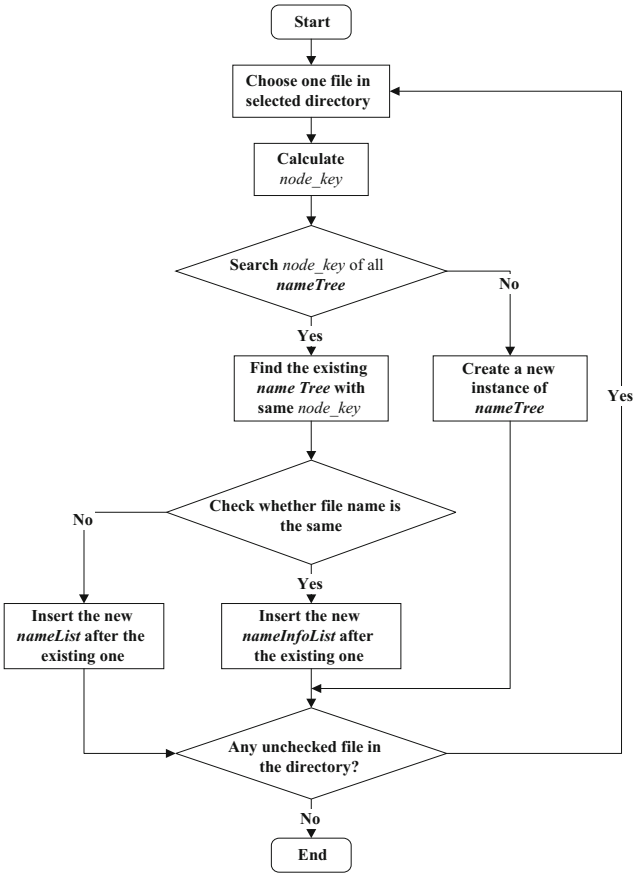

Fig. 6 The processing flow in the filename based approach

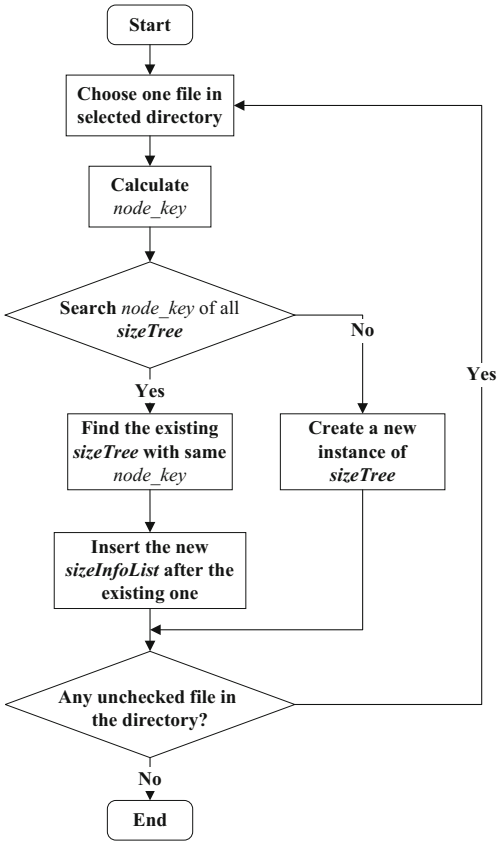

Fig. 7 The processing flow in the file size based approach

node_key. Once the *sizeTree* with the same *node_key* is found, the process inserts the new *sizeInfoList* after the one in the existing *sizeTree*, and changes 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 creates a new instance of *sizeTree* and store the file information into *nameList* and *nameInfoList*. Subsequently, the process loops until there is no unchecked file in the selected directory. The entire processing flow is shown in Fig. 7.

3) Find the same MD5 hash value among files

Since the approach is an extension of file size based approach, the processing flow of the MD5 hash based approach is similar to the file size based approach. Once the *sizeTree* with same *node_key* is found, the process calculates *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 there is a match, the process inserts the new *sizeInfoList* after the one in the existing *sizeTree*, and changes the value of *is_dup* to note that the duplicated file with the same MD5 hash value is found. If no match founds, the process takes 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*. The entire processing flow is shown in Fig. 8

C 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 scenario of the time complexity of the second part may be $\frac{N(N+1)}{2}=O\left(N^2\right)$, if hash values of all files need to be calculated.

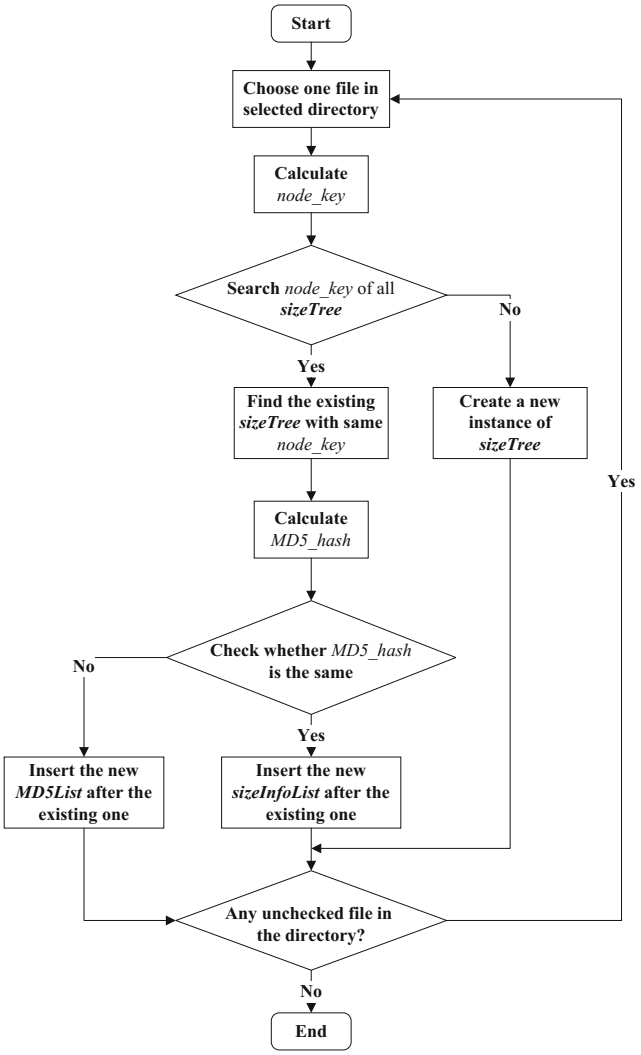

Fig. 8 The processing flow in the hash value based approach

4 Theoretical analysis and qualitative analysis

In this section, we analyze the average-case time complexity of a linear based scheme and a hash based lookup scheme from a probabilistic viewpoint [14].

4.1 General average-case complexity analysis about time cost

First, we introduce how to conduct a general average-case time complexity analysis for a scheme. Assume that a scheme S with a set D of inputs such that S has the definiteness and stop properties on D. Let W be a set of sizes of inputs and let $|\cdot|$ be a size function of inputs in D. Let $t(d)$ be the time cost function for S. Meanwhile, the following two conditions hold for each size w in W:

- The set $\{d \in D : |d| = w\}$ is finite;
- A probability function is defined on the set of inputs of size w and is denoted by P_w

By definition,

$$\sum_{|d|=w} P_w = 1 \tag{1}$$

For each size w in W the restriction of the time cost function $t(\cdot)$ to inputs of size w is a random variable; it is denoted by τw. The random variable τw assumes natural numbers are values. The probability distribution of τw is denoted by p_{wk}, that is, the probability that for an input d of size w, τw is equal to k. Notice that the average time complexity is

$$T_{ave}(w) = \sum_{|d|=w} P_w(d)\cdot t(d) = \sum_{k\geq 0} k\left(\sum_{|d|=w,t(d)=k} p_w(d)\right) = \sum_{k\geq 0} kp_{wk} \tag{2}$$

For each size w in W, that is, the average time complexity of S for input size w is equal to the mean value $E(w)$ of the random variable τw. Apart from the function $E(w)$ the statistical properties of the running time of S are also characterized by the variance function $V(w)$ and standard deviation $D(w)$ of τw with w ranging over W, where

$$\mathrm{V}(w) = \sum_{k\geq 0} (k\text{-}T_{ave}(w))^2 p_{wk} \tag{3}$$

and

$$\mathrm{D}(w) = \sqrt{\mathrm{V}(w)} \tag{4}$$

These quantities determine how much the random variables τw are concentrated around their mean values. The smaller the standard deviation the better concentration of τw around its mean value is. To find the statistical quantities $E(w)$, $V(w)$, and $D(w)$, the method of generating function is used. The generating function for random variables τw is

$$P_w(z) = \sum_{k\geq 0} \mathrm{p}_{wk} z^k \tag{5}$$

with arguments and values being real numbers. Therefore,

$$\mathrm{P}'_{\mathrm{w}}(1) = \left(\sum_{k\geq 0} \mathrm{kp}_{wk} z^{k\text{-}1}\right)(1) = \sum_{k\geq 0} kp_{wk} \tag{6}$$

By (2) and (6), we can get

$$\mathrm{E(w)} = \mathrm{P}'_{\mathrm{w}}(1) \tag{7}$$

Next

$$\mathrm{P}''_{\mathrm{w}}(1) = \sum_{k\geq 0} \mathrm{k(k\text{-}1)p}_{wk} z^{k\text{-}2}(1) = \sum_{k\geq 0} k(k\text{-}1)p_{wk} \tag{8}$$

By (2), (3) and (8), we can get

$$\mathrm{V}(w) = \sum_{k\geq 0} \left(k\text{-}\mathrm{P}'_{\mathrm{w}}(1)\right)^2 p_{wk} = \mathrm{P}''_{\mathrm{w}}(1) + \mathrm{P}'_{\mathrm{w}}(1)\text{-}\mathrm{P}'_{\mathrm{w}}(1)^2 \tag{9}$$

4.2 The time complexity analysis about the hash based lookup scheme

Base on previous subsection, assume an array $H[0..M-1]$ is used to store the elements of A and a hash function $h: A \rightarrow \{0, 1, \ldots, M-1\}$ is used to associate an item of H with an element x of A. The computation of $h(x)$ should be very fast and h should distribute the elements of A as uniformly as possible. For a given x, if position $H[h(x)]$ is empty, it means that $x \notin A$ and x can be stored at this position. If $H[h(x)]$ is occupied, then either $H[h(x)] = x$ or $H[h(x)] \neq x$. In the first case, the lookup process ends with the result true; in the second, the collision problem happens. A common solution to the collision problem is separate chaining to maintain in each entry $H[h(x)]$, $i = 0, 1, \ldots, M-1$, a linear chain of elements $x \in A$ such that $h(x) = i$. The probabilistic analysis of separate chaining is shown as follows. We start this analysis from the assumption that for $A = \{x_1, \ldots, x_n\}$ the corresponding sequence of hash codes, $h_1 = h(x_1), \ldots, h_n = h(x_n)$, has the probability of occurrence $1/M^n$, that is, the probability that $h_i = j$ for $1 \leq i \leq n$ and $0 \leq j < M$ is $1/M$.

Let us consider $T^-(n)$. Assume that $h(x) = j$ when an unsuccessful search for x happens. Denoting by p_{nk} the probability that the list $H[j]$ has length k we have

$$p_{nk} = \binom{n}{k} \left(\frac{1}{M}\right)^k \left(1-\frac{1}{M}\right)^{n-k} \tag{10}$$

Since the value $j$appears k times in a sequence $h_1, \ldots, h_n$ with the probability defined by the Bernoulli schema, now we have

$$P_n^-(z) = \sum_{k\geq 0} p_{nk} z^{k+1} \tag{11}$$

which can easily be transformed to a simpler form

$$P_n^-(z) = \left(\frac{z}{M} + 1 - \frac{1}{M}\right)^n z \tag{12}$$

By differentiating (12), we have

$$P_n^{-\prime}(z) = \left(\frac{z}{M} + 1 - \frac{1}{M}\right)^{n-1} \left(\frac{nz}{M} + \frac{z}{M} + 1 - \frac{1}{M}\right) \tag{13}$$

$$P_n^{-\prime\prime}(z) = \left(\frac{z}{M} + 1 - \frac{1}{M}\right)^{n-2} \left(\frac{n-1}{M}\left(\frac{nz}{M} + \frac{z}{M} + 1 - \frac{1}{M}\right) + \left(\frac{z}{M} + 1 - \frac{1}{M}\right)\left(\frac{n}{M} + \frac{1}{M}\right)\right) \tag{14}$$

By (13) and (14), we can get

$$P_n^{-\prime}(1) = \frac{n}{M} + 1 \tag{15}$$

$$P_n^{-\prime\prime}(1) = \frac{n(n-1)}{M^2} + \frac{2n}{M} \tag{16}$$

$$E^-(n) = P_n^{-\prime}(1) = \frac{n}{M} + 1 \tag{17}$$

$$\begin{aligned} V^-(n) &= P_n^{-\prime\prime}(1) + P_n^{-\prime}(1) - \left(P_n^{-\prime}(1)\right)^2 \\ &= \frac{n(n-1)}{M^2} + \frac{2n}{M} + \left(\frac{n}{M} + 1\right) - \left(\frac{n}{M} + 1\right)^2 \\ &= \frac{n(M-1)}{M^2} \end{aligned} \tag{18}$$

and denoting by $\alpha = n/M$, that is, the definition of table H, we obtain

$$E^-(n) = \alpha + 1,\ V^-(n) \cong \alpha,\ D^-(n) \cong \sqrt{\alpha}, \tag{19}$$

To estimate $E^+(n)$, that is, the average cost in a successful case, consider the function $nE^+(n)$. Its value is equal to the total number of steps performed when all n elements of A are searched for. But the list of length k contributes $\frac{1}{2}k(k+1)$ steps to the total. Consequently, since there are M lists, we have

$$nE^+(n) = M \sum_{k\geq 0} \frac{k(k+1)}{2} p_{nk} \tag{20}$$

and according to the definition of $P_n^-(z)$ we obtain

$$E^+(n) = \frac{M}{2n} \sum_{k\geq 0} k(k+1) p_{nk} = \frac{M}{2n} P_n^{-\prime\prime}(1) = \frac{n-1}{2M} + 1 \cong \frac{1}{2}\alpha + 1 \tag{21}$$

4.3 Qualitative analysis

Followed the analysis of the scheme from previous sub-section using hash function values as indexing mechanism searching for a matching file content and determine which part of the file can be used as *representative block*, has a generic lookup time complexty of $O(1)$, and worst case time complexity of $O(n)$, if the original files are chopped into n blocks. The storage overhead for these blocks and hash values is $O(n)$ + hash table size.

It is prudent to apply a cryptographic hash function as the index [15]. Such function has the property such that it is computationally infeasible to find two distinct files/blocks with the same value, and preserve the assumption that files/blocks hash values can be used as file fingerprints. The MD5 hash has a

Table 1 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

length of 128 bit (16 bytes) hash value, usually represents as 32 hex numbers. The calculation of MD5 hash can be extremely fast at the level of hundreds of MB per seconds even for mobile devices that has limited computational resources. And assuming its hash value is spread in a uniform distribution, the probability of the event that one or more collisions occur is bounded by

$$p \leq \frac{m(m\text{-}1)}{2} \cdot 2^{-128} \tag{22}$$

Assuming a collection of m files in a 10TB level private cloud secondary storage NAS device, even consider every file/block to be the minimum size of 1KB. The number of unique files could be as many as 10^2, the probability of a collision should be smaller than 10^{-18}. And since most multimedia files shared in private cloud storage are much larger than 1KB, the collision probability using MD5 as hash value function for file deduplication index is neglectable. However as storage capacity continues to grow each year, a more collision resistant hash function, like SHA family cryptographic hash functions with more than 160 bit (20 bytes) digits long hash value, could be used in the future, and also make cross cloud storage indexing scheme secure.

A secure cryptographic hash function can also function as an unique handle when deduplications needed to be performed across multiple storage systems, in both inline or post processing schemes.

5 Evaluation and experiment results

5.1 Evaluation environment

To evaluate the performance of implemented schemes, we used a typical network-attached storage (NAS) device as the storage device to run four deduplication check schemes described in Section 3. The NAS used in the evaluation is QNAP NAS TS-269L, and its specification is shown in Table 1.

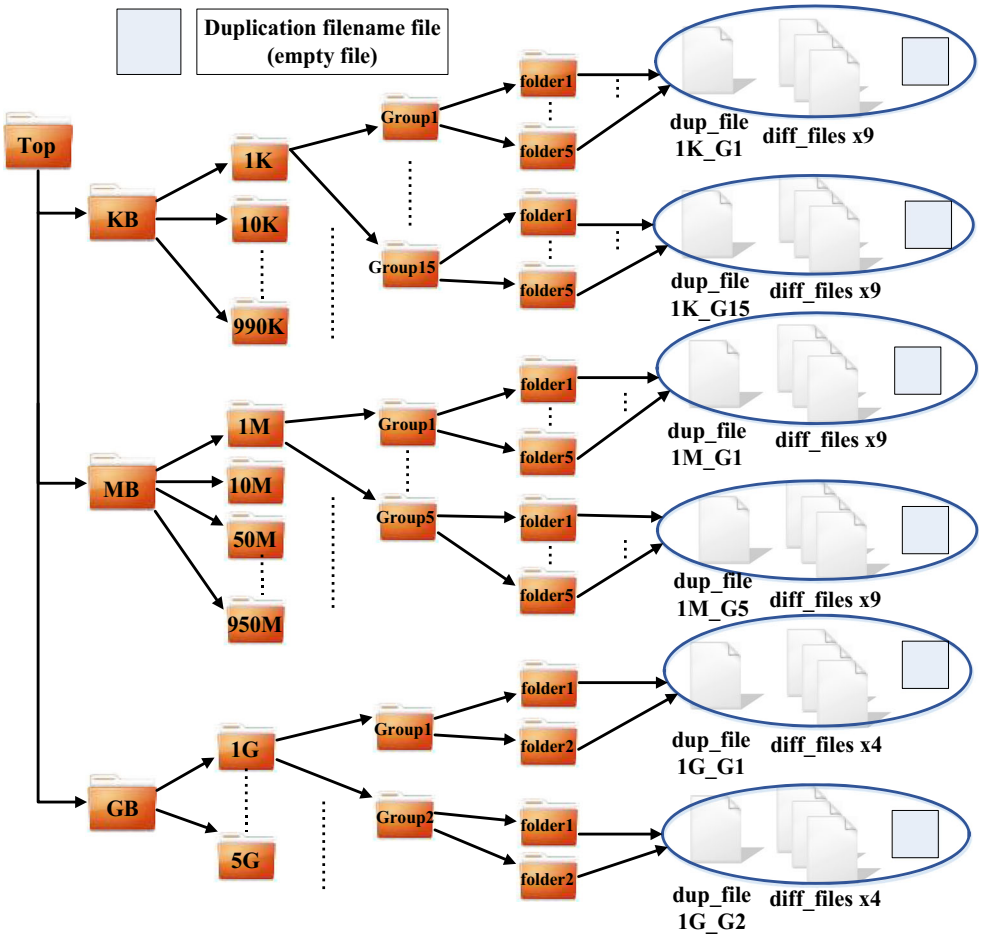

Fig. 9 The directory tree diagram of of the testing scenario

Table 2 Time evaluation for each scenario

	Filename	File Size	MD5 (Partial)	MD5 (Full)
KB	1m13s	1m16s	3m52s	3m52s
MB	5.2s	6.6s	41.4s	43m49s
GB	0.53s	0.64s	47.6s	11m33s

In the evaluation, we design three different testing scenarios to test these four 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. 9.

From the study of [7], the distribution of individual file sizes can range from as small as several KB, up to 4GB, where half of all bytes are in files larger than 30MB. Where smaller size file have more duplications than larger files, while the distribution of number of different size files follows, where larger files are mostly exact copy of each other if they are duplicated with fewer copies. The setup of each scenario in our evaluation experiments are as follows.

1) KB scenario, the total file size of the first sub-folder 1K is 1 kilobyte, the second sub-folder 10K is 10 kilobytes, and so on. Respectively, the total file size of each under the i sub-folders is $S_{kb} = 10i$, and the values of i are ranged from 1 to 99. There are 15 group-folders in each sub-folder, and each group-folder has 5 end-folders. In each end-folder, there are 10 files with 9 different files and 1 duplicated file from another end-folder, and 1 fixed duplicated filename file. The total file amount in the scenario is 82,500.
2) MB scenario, the total file size of the first sub-folder 1M is 1 megabyte, and the second sub-folder 10M is 10 megabytes. 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 are 9 different files, 1 duplicated file, and 1 fixed duplicated filename file. The total file amount in the scenario is 5775.
3) GB scenario, the total file size of each sub-folder k is $S_{gb} = k$ gigabytes, and the value of u 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 4 different files, a duplication file, and a duplicated filename file. The total file amount in the scenario is 120.

Table 3 CPU usage for each scenario

	File Name	File Size	MD5 (Partial)	MD5 (Full)
KB	8.2%	10.7%	12.2%	12.3%
MB	2.4%	6.7%	21.0%	24.9%
GB	neglectable	neglectable	24.7%	24.9%

Table 4 Memory usage for each scenario

	File Name	File Size	MD5 (Partial)	MD5 (Full)
KB	5.8% (57MB)	5.8% (58MB)	8.0% (79MB)	8.0% (79MB)
MB	0.1% (1384B)	0.4% (4292B)	0.6% (6488B)	0.6% (6476B)
GB	neglectable	neglectable	<1% (884B)	<1% (888B)

5.2 Evaluation results

The overall combined evaluation results of three scenarios using four file deduplication schemes are shown in Tables 2, 3, 4, and 5. In Table 2, "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 as representative blocks to calculate the MD5 hash value, while MD5 (Full) uses the complete file content to calculate the MD5 hash value, always assume to be in worst case.

Table 2 shows the experiment result of running time of each scenario. It shows using partial file as representative block for larger files to calculate hash value is indeed within reasonable running time, and significantly faster on the magnitude of 10 folds to 100 folds. While in KB sized files almost all file contents have to be used to generate hash value file fingerprint.

Table 3 shows the experiment result of CPU usage of each scenario. It shows using MD5 as hash function to calculate each file/block fingerprint is fast enough, and doesn't cost significant amount of computational resources in each case.

Table 4 shows the experiment result of memory usage of each scenario. It verifies that very limited memory space is needed during MD5 hash value calculation..

Finally in Table 5, we observe that as expected the filename based approach and the file size based approach perform faster compared to MD5 hash value based ones. And worst case scenario (or safer option) as the time complexity of full file MD5 hash value based approach required much more

Table 5 Overall 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)

computational resources and significant slower than partial one. 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. However, using partials of larger files as representative blocks to calculate MD5 hash value performs only slightly worse than simple filename or file size based approaches, and can be considered as a reasonable trade-off between accuracy and efficiency for a post processing deduplication scheme.

From the results above, we can see that the CPU usage of each scenario is below 25%, where in partial MD5 scheme the weight average of CPU usage based on file size is 19.3%, while the full MD5 scheme is 20.7%. The CPU usage is limited due to the pipeline process in MD5.

Where compare the total computational time, the partial MD5 scheme is 10 times faster than full MD5 scheme. Since MD5 is several magnitude faster than other more recent cryptography hash algorithm, such as SHA family, It can be safely assume that implementing with more complex cryptography algorithm would considerable slow down the deduplication process, even when just using partial file as representative block as fingerprint.

But the most constraint resource in a private cloud storage appliance, such as a NAS device is the memory. Since it is not built to run GUI interface, and usually has a very limited memory to work with. Our schemes all use very little memory less than 100MB. This is quite crucial as a background process or to be run manually while the NAS appliance has other applications need to run in parallel, and serves its main function as cloud storage device.

6 Conclusion

Implementing effective file deduplication schemes running in small private cloud storages are crucial in balancing network bandwidth usage, local computational resources, and the utilization of storage space. In this paper, we have introduced and verified several deduplication schemes to test their viability. The experimental results show the use of partial file based hash value scheme is successful in meeting the criteria. The network web based API user interface for these schemes implemented on NAS devices is practical and functional for daily use. Bridging the current gap of deduplication appliances in small private cloud storage services, and connected the mixed used of many public cloud storage services parallel with multiple sources of data generated by uncoordinated users.

The hash function based file fingerprint indexing scheme also has the potential to be expanded into a cross platform cloud storage deduplication and data sharing, searching, management service in the future.

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

References

1. Tate J, Beck P, Ibarra HH, Kumaravel S, Miklas L (2012) Introduction to storage area networks and system networking. IBM Redbooks
2. Hong B, Plantenberg D, Long DD, & Sivan-Zimet M (2004) "Duplicate Data Elimination in a SAN File System". In MSST (pp. 301–314)
3. Bobbarjung DR, Jagannathan S, Dubnicki C (2006) Improving duplicate elimination in storage systems. ACM Trans Storage (TOS) 2(4):424–448
4. Min J, Yoon D, Won Y (2011) Efficient deduplication techniques for modern backup operation. Comput IEEE Trans on 60(6):824–840
5. Li J, Li YK, Chen X, Lee PP, Lou W (2015) A hybrid cloud approach for secure authorized deduplication. Parallel and Distrib Sys IEEE Trans on 26(5):1206–1216
6. Stanek J, Sorniotti A, Androulaki E, Kencl L (2014) A secure data deduplication scheme for cloud storage in financial cryptography and data security. Springer, Berlin Heidelberg, pp 99–118
7. Meyer DT, Bolosky WJ (2012) A study of practical deduplication. ACM Trans Storage (TOS) 7(4):14
8. Harnik D, Pinkas B, Shulman-Peleg A (2010) Side channels in cloud services: deduplication in cloud storage. Security & Privacy IEEE 8(6):40–47
9. Paulo J, Pereira J (2014) A survey and classification of storage deduplication systems. ACM Comput Surveys (CSUR) 47(1):11
10. Meister D., & Brinkmann A (2009) Multi-level comparison of data deduplication in a backup scenario. In Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference (p. 8). ACM
11. Henson V (2003) An Analysis of Compare-by-hash. In HotOS (pp. 13–18)
12. Malhotra J, & Bakal J (2015) A survey and comparative study of data deduplication techniques. In Pervasive Computing (ICPC), 2015 International Conference on (pp. 1–5). IEEE
13. Rivest R (1992) The MD5 message-digest algorithm. RFC 1321
14. Banachowski L, Kreczmar A, Rytter W (1991) Analysis of Algorithms and Data Structures
15. Quinlan S, & Dorward S (2002) Venti: A New Approach to Archival Storage. In *FAST* (Vol. 2, pp. 89–101).2