Front. Optoelectron. China 2010, 3(3): 308–316
DOI 10.1007/s12200-010-0103-z

RESEARCH ARTICLE

Optimization for data de-duplication algorithm based on file content

Xuejun NIE, Leihua QIN (✉), Jingli ZHOU, Ke LIU, Jianfeng ZHU, Yu WANG

School of Computer Science and Technology, Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan 430074, China

© Higher Education Press and Springer-Verlag Berlin Heidelberg 2010

Abstract Content defined chunking (CDC) is a prevalent data de-duplication algorithm for removing redundant data segments in archival storage systems. Current researches on CDC do not consider the unique content characteristic of different file types, and they determine chunk boundaries in a random way and apply a single strategy for all file types. It has been proven that such method cannot achieve optimal performance for compound archival data. We analyze the content characteristic of different file types and propose candidate anchor histogram (CAH) to capture it. We propose an improved strategy for determining chunk boundaries based on CAH and tune some key parameters of CDC based on the data layout of underlying data de-duplication file system (TriDFS), which can efficiently store variable-sized chunks on fixed-sized physical blocks. These strategies are evaluated with representative archival data, and the result indicates that they can increase on average the compression ratio by 16.3% and write throughput by 13.7%, while only decrease the read throughput by 2.5%.

Keywords data de-duplication, content defined chunking (CDC), file content, candidate anchor histogram (CAH)

1 Introduction

With the explosion in disk capacity, replacing slow tapes with high-speed disk can enable organizations to increase backup/recovery performance in archival storage systems. Because of the fact that a large proportion of data does not change between backup sessions, there is massive redundancy in archival sets [1,2]. De-duplication is an effective way to minimize the amount of archival data that must be stored and managed, and it is practical only with random-access devices (such as disk) instead of sequential-access devices (such as tape) [3].

One of the crucial prerequisites in de-duplication is to identify duplicate data segments. Early de-duplication storage systems [4,5] partition files into fixed-sized data chunks (the chunk size is always equal to the physical block size of storage devices), and identify duplicate chunks by comparing their fingerprints, which are hash keys computed by some hash function. This approach is independent of the content of the files being stored. As we would expect, the effectiveness of this approach on duplicate elimination is highly sensitive to the sequence of edits and modifications performed on consecutive versions of a file. A single addition, deletion or insertion can easily shift the remaining content. To tolerate shifted contents, modern de-duplication systems partition files into variable-size chunks based on their contents. This approach is also referred to as content defined chunking (CDC) [6–8]. You et al. [9] have demonstrated that CDC can achieve much higher compression ratio than fixed-sized approaches. However, CDC also brings some new challenges, one of which is how to determine the boundaries of variable-sized chunks. Manber has proposed a method of choosing a fixed number of bytes that satisfy specific conditions as chunk boundary, which are also referred to as an anchor [10]. Manber has shown that Rabin fingerprint [11] is an efficient algorithm to compute the anchor.

However, the current CDC algorithm does not consider the unique content characteristic of different file types. When choosing anchors, current CDC [6–8,10,12] assumes that the distribution of byte value in the byte stream of any file type conforms to a uniform distribution on the interval [0,255]. This assumption is true for some file types, such as RGB image file, where each pixel consists of three bytes and the value range of each byte is [0,255]. However, other file types do not conform to such

Received February 26, 2010; accepted April 6, 2010

E-mail: lhqin@mail.hust.edu.cn

uniform distribution. For example, the value range of most bytes in C source code files is [32,126] instead of [0,255], because all these bytes are visible characters. Cox, Manber, et al. [4,10] have also pointed out that adopting the strategy of determining anchors to different file types is a promising way to further improve compression ratio of CDC.

In this paper, we propose candidate anchor histogram (CAH) to capture the content characteristic (that is, the distribution of byte value) of a file type. We have proven that there is a one-to-one mapping between CAH and the distribution of byte value. We detail a strategy of determining anchors based on CAH and prove that it can achieve optimal or nearly optimal compression ratio for CDC. Besides that, we implement a data de-duplication file system (TriDFS) to accommodate the variable-sized chunks generated by CDC, and tune some key parameters of CDC according to the data layout on TriDFS.

To evaluate these techniques, we choose several representative archival data sets to simulate the real archival workloads. As the result shows, these techniques can increase on average the compression ratio by 16.3% and write throughput by 13.7%, respectively. However, due to the overheads introduced by extra computing, the read throughput decreases on average by 2.5%.

The rest of the paper is organized as follows. Section 2 shows the challenges in current CDC; Sect. 3 describes our methods of dealing with these challenges; Sect. 4 shows our evaluation result, and Sect. 5 draws conclusion and plans some future work.

2 Challenges in current CDC de-duplication

2.1 Single strategy for all file types

The basic CDC algorithm can be described as follows:

a) Given a file d, first choose a slide window of w bytes in length.

b) Move the slide window forward in the byte stream of d, every time by an offset of a fixed number of bytes; Compute a checksum with the w bytes in the slide window. This checksum is also referred to as a fingerprint, denoted as f.

c) If f satisfies some predefined criteria, such as when the result that f modulo some integer divisor is zero, then the w bytes in the current slide window is considered as a chunk boundary, which is also referred to as an anchor, denoted as a.

d) All bytes between two anchors are considered as a chunk.

e) By moving the slide window throughout the whole byte stream of d, the file is partitioned into a set of variable-sized chunks.

The strategy of determining whether f is an anchor can be described as follows: Randomly choose a magic value m, where the value range of m is $[0,2^k-1]$; If the last k bits of f is equal to m, then the w bytes with which f is computed is considered as an anchor. All anchors based on m constitute an anchor set, denoted as $A_m = \{f_{m1}, f_{m2},...,f_{mn}\}$.

In the above strategy, there are two debatable points. One is choosing the magic value m in a random way, and the other is determining anchors for all file types with the single m. There is an implicit assumption within these points: the byte value in the byte stream of different file types conforms to the same uniform distribution on the interval [0,255]. Because of such assumption, for any magic value m that is randomly chosen in the range of $[0,2^k-1]$, the number of anchors determined by m is equal to the number of anchors determined by any other value in $[0,2^k-1]$. Clearly, this strategy of choosing m is independent to the distribution of byte value in the file.

This implicit assumption is true for some file types, but not for all. For example, in C source code file with ASCII encoding, the value range of most bytes is [32,126] instead of [0,255], because all these bytes are visible characters and invisible or unreadable characters are rarely used in writing programs. Similarly, with Unicode-16 encoding, every character is encoded in two bytes, and there is a relationship between the first byte and the second byte. Therefore, the byte values are actually not independent of each other.

According to the above analyses, we can conclude that the byte value distributions in different file types do not conform to the same uniform distribution on the interval [0,255]. As the compression ratio is tightly related with the anchors determined by magic value, choosing the magic value in a random way may not achieve optimal performance for all file types.

We refer to this difference in byte value distribution as a unique content characteristic of a file type. As a rule of thumb, if we add content-specific factors to CDC, its adaptability to different file types can be improved, and so does the compression ratio.

2.2 Store variable-sized chuck on fixed-sized physical block

Traditionally, physical blocks on which data are stored can always have a fixed size. Usually, there are two ways in storing variable-sized chunks. The first way is referred to as Padding. When storing chunks in a padding way, the last block of chunk may not be filled up and the remainder of the block will be padded with zero. Padding obviates the need to maintain extra information for each chunk, but suffers from internal fragmentation that can consume lots of unnecessary space. The second way is referred to as Packing. When storing chunks in a packing way, variable-sized chunks are packed contiguously into a lager and fixed-sized storage unit, which is always consisted of a

group of continuous physical blocks. This method can effectively reduce padding space; however, it has to maintain the physical blocks number and offset where each chunk begins.

In this paper, we take the second way and implement a TriDFS. The storage unit in TriDFS is referred to as a container, which consists of a fixed number of continuous physical blocks. Therefore, when calculating storage utilization, we should consider the bytes that are required to maintain extra information for each chunk.

3 Methods

3.1 Candidate anchor histogram (CAH)

From previous analyses, we expect that by choosing the magic value based on the byte value distribution and applying it to all files with similar distribution, we can identify more redundancy among these files. We propose the concept of CAH to represent the byte value distribution.

Definition 1 Candidate anchor

Given a magic value m, where the value range of m is $[0,2^k-1]$. Denote the w bytes in slide window as $(t_1,t_2,...,t_w)$, and compute the fingerprint f with Rabin fingerprint; If the last k bits of f is equal to m, then the w bytes with which f is computed is considered as a candidate anchor, denoted as a_m;

All candidate anchors determined by m constitute a candidate anchor set, denoted as $C_m = \{a_{m1},a_{m2},...,a_{mn}\}$.

Definition 2 CAH

Denote the number of elements in C_m as P_m, $p_m = |C_m|$. For each value i in $[0,2^k-1]$, there is always a p_i, so the CAH is defined as $F = \{p_0,p_1,...,p_{2^k-1}\}$.

Definition 2.1 CAH of a single file

For a single file d, its CAH is just the same as the CAH in Definition 2, denoted as F_d.

Definition 2.2 CAH of a file type

For a file type, first choose a sample file set of that type, denoted as D; then compute the CAH of each single file. After that, add up the CAH of all files, and the result is the CAH of that file type, denoted as follows:

$$F_D = \sum_{i=0}^{n} F_{di} = \{P_0,P_1,...P_{2^k-1}\}.$$

Theorem 1 The relationship between CAH and the distribution of byte value is a one-to-one mapping.

Proof a) The distribution of byte value can be captured as a histogram, denoted as $H = \{b_0,b_1,...,b_{255}\}$, where b_i is the number of bytes whose value is equal to i.

b) By Rabin fingerprint, the probability of fingerprint collision (that is, the fingerprints of different byte sequences are computed to be equal) is 2^{-w}.

c) Given a large storage system that may contain a petabyte (10^{15} bytes) of data, as long as $w > \log_2 10^{15} = 49.829$, that is, if we let $w = 50$, the relationship between byte sequence and its fingerprint is a one-to-one mapping. For larger archival storage systems, we can choose a larger value for w.

d) CAH is computed by counting the number of anchors determined by some magic value m. Each anchor set represents a group of unique byte sequences, so counting the number of anchors is the same as counting the number of byte sequences.

e) By c), d) and transitivity, we can deduce that the relationship between CAH and the distribution of byte value is a one-to-one mapping.

CAH can be illustrated with general histograms. In the histogram, the x-axis represents the range of $[0,2^k-1]$ and the y-axis represents each p_i in F_d or P_i in F_D. In Fig. 1, we show the CAHs for single file and file type respectively, where $k = 10$. There are three kinds of files: html, email and mp3. For each kind, we show both the F_D of the file type and the F_d of a single file. From these figures, we can obtain some observations:

a) Different file types have different CAHs. Indeed, the CAH of mp3 file conforms to the uniform distribution of [0,255], however other file types do not. Therefore, we should not apply a single strategy for all file types when choosing anchors.

b) The CAH of a single file is very similar to the CAH of the same file type. This observation answers our second question. We can group files by their type, specifically by their extension. With those observations, we can up front choose a sample file set of the same type, compute the optimal magic value m for this file type, and then apply it for all files of that type.

3.2 Choose magic value based on CAH

One of the key parameters of CDC is the average size of all chunks, denoted as S_{ave}. The chunk sizes, although variable, are expected to be within a margin of error of S_{ave}. By original assumption that byte value in a data stream of any file type conforms to uniform distribution on the interval [0,255], S_{ave} is determined by the number of last bits of fingerprint that we choose to compare with the magic value, that is, $S_{\text{ave}} = 2^k$. For example, if we let $k = 13$, then the average chunk size is $2^{13} = 8$ kB. However, due to the uneven distribution of byte value in different file types, if we choose the magic value in a random way, the average chunk size may not be equal to 2^k.

The average chunk size is a key parameter because of its impact on compression and performance [7–9]. Therefore, it is important to make sure that the average size of all generated chunks is equal to or nearly equal to S_{ave}. The key of making this guarantee is to choose the magic value carefully. With CAH, we can choose a magic value that can

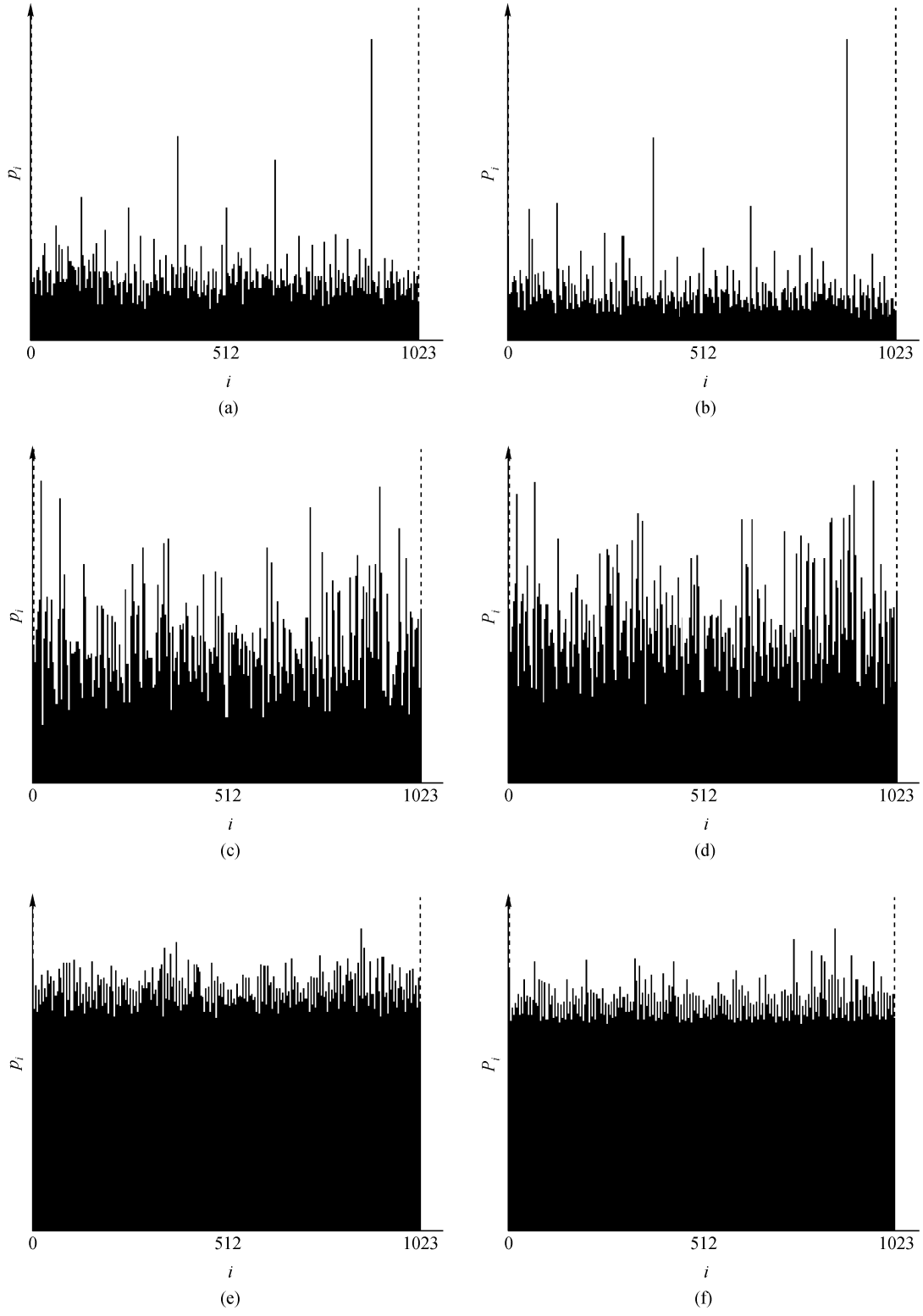


Fig. 1 F_d and F_D for file type of html, email and mp3, respectively. (a) F_d of 1 html file; (b) F_D of 2127 html files; (c) F_d of 1 email file; (d) F_D of 300 email files; (e) F_d of 1 mp3 file; (f) F_D of 87 mp3 files

generate chunks whose average size is nearly equal to S_{ave}. That is, given a S_{ave} and a set of file, we can first compute the expected number of chunks that these files should be partitioned into, then choose the magic value by which the number of chunks is determined is closest to the expected number. The whole algorithm of choosing the magic value based on CAH is shown in Fig. 2.

While CAH can identify more redundancy among files of the same type, it has lost the redundancy among different file types. Only if the redundancy among different file types is far smaller than that among files of the same type, can CAH improve the overall compression ratio.

3.3 TriDFS

We implement a TriDFS to store variable-sized chunks. TriDFS is a micro file system, which is extended based on EXT3. In TriDFS, the i-node (index node) of each file consists of a set of hash keys instead of original logical block number (LBN), and each hash key is computed by SHA-1 algorithm. Variable-sized chunks are packed into a relatively large and fixed-sized storage unit, which we referred to as a container. All hash keys are recorded in a hash table. Each item in the table contains information of an unduplicated chunk, including the chunk's hash key, size, reference count (which indicates the duplicate count of that chunk and when it is 0 the chunk should be removed from the file system), the container number in which the chunk is stored and the offset of the chunk in that container. The logical structure of i-node, hash table, container and the addressing process are shown in Fig. 3.

TriDFS has defined several properties for its data layout, including S_C (the total size of a container), N_C (the maximum number of chunks that a container can store), S_{offset} (the size of an offset in container), and S_{meta} (the size of the metadata for each chunk). Among these properties,

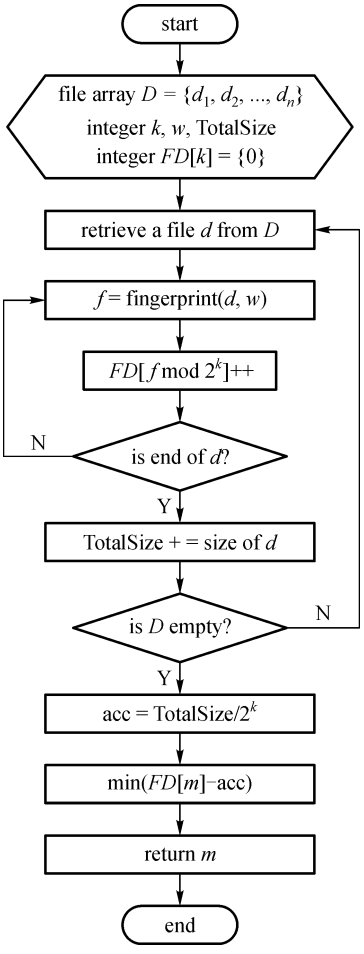


Fig. 2 Choose magic value algorithm

S_{offset} and S_{meta} are constant values, which are 2 bytes and 16 bytes, respectively; S_C is also a constant value, because it is equal to the size of the cache line between the memory and disk. In our TriDFS, S_C is set to 64 kB. However, N_C is

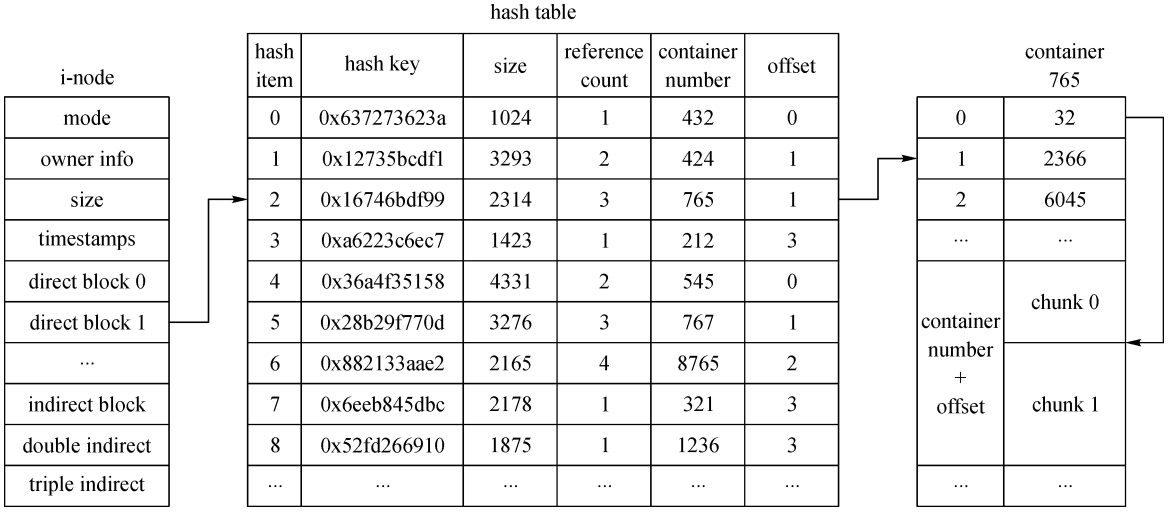


Fig. 3 Logical structure of i-node, hash table, container and addressing process

the only property that can be configured or tuned by the administrator, it indicates the maximum number of chunks that can be stored in a container. To improve the storage utilization, we should make sure that chunk sizes match TriDFS's data layout as well as possible. Therefore, we should tune some parameters of CDC based on the above properties.

First of all, the S_{ave} of CDC can be determined as follows:

$$S_{ave} = (S_C - N_C \times S_{offset})/N_C.$$

To deal with ill conditions, that is, to prevent chunks from being too long or too short, we should set two limits: the maximum chunk size S_{max} and the minimum chunk size S_{min}. Apparently, S_{max} should not exceed the maximum valid space in a container. Therefore, S_{max} is determined as follows:

$$S_{max} = S_C - N_C \times S_{offset}.$$

The minimum chunk size S_{min} must be greater than the size of metadata that maintain extra information for each chunk, otherwise, CDC can be counterproductive. However, if S_{min} is set with an unnecessarily large value, it will have a negative impact on the compression ration. After several iterative design processes, we have chosen to use 256 for S_{min}:

$$S_{min} = 256.$$

The last parameter that need to be set is the slide window size w. As we have analyzed before, w is tightly associated with the total storage capacity. In our evaluation, we assume that the maximum capacity of a storage system is 1 PB, then

$$w = 50 > \log_2 10^{15}.$$

4 Evaluation

This paper mainly focuses on two evaluation aspects: the compression ratio achieved by CAH and TriDFS and performance variation caused by the extra overhead introduced by these techniques.

Our de-duplication storage system is deployed as an internet protocol (IP) storage area network (SAN), and iSCSI protocol is used as the data transport protocol. The computer used as iSCSI initiator is equipped with an AMD dual-core processor at 2.21 GHz, 1 GB random-access memory (RAM), and 160 GB disk. The computer used as iSCSI target is equipped with an Intel Pentium 4 processor at 2.4 GHz, 1.2 GB RAM, and 160 GB disk. The two computers are connected with 1 Gb/s independent Ethernet network. TriDFS is installed on the target computer, and exposes standard file system interface of EXT3 to the initiator.

4.1 Workload characteristic

We chose 4 representative data sets to simulate real archival workloads, and the file types of these data sets are email message, source code, web page and audio. Email messages often contain headers and attachments that show great resemblance. Source code and web content are typically versioned. Non-textual content such as audios are often similar and require lots of storage space. The details of each data set are described in Table 1.

Table 1 Details of 4 data sets

data set		description
1	title	Linux kernel 2.6 Source code
	from	http://www.kernel.org
	detail	7 versions (from 2.6.12 to 2.6.18), including 131596 files, and the total size is 4902.0 MB.
2	title	web page data for TREC 2008
	from	http://es.csiro.au/cerc/data
	detail	web pages gathered from CSIRO (http://www.csiro.au), including 267 subset (000-266), 409551 files, and the total size is 4499.8 MB.
3	title	emails data for TREC 2005
	from	http://plg.uwaterloo.ca/~gvcormac/treccorpus/
	detail	emails gathered from CSIRO, including 307 subsets (CSIRO000-CSIRO306), 92189 emails, and the total size is 3464.5 MB.
4	title	mp3 files
	from	http://www.diggcd.com/
	detail	mp3 files downloaded from music site of Huazhong University of Science and Technology (HUST), including 1005 mp3 files, and the total size is 4614.5 MB.

4.2 Compression ratio

According to the chosen magic value algorithm, we should first choose a sample subset from each data set. The sample subsets are described in Table 2.

Table 2 Sample subsets

data set	sample subset
1	all files under the directory of Driver in 2.6.12, including 4360 files, and the total size is 192.3 MB
2	subset CSIRO000, including 2127 files, and the total size is 192.3 MB
3	subset 000, including 300 files, and the total size is 9.5 MB
4	6 collections, including 87 files, and the total size is 490.1 MB

Next, we compute the optimal magic value for each sample subset. With previous analyses, we know that N_C is the only property of TriDFS that can be configured by the administrator. For each value of N_C, there is a

corresponding S_{ave} and it is the most crucial parameter of CDC. Therefore, we choose S_{ave} as the only independent variable, and compute magic values based on different S_{ave}. The results are shown in Table 3.

Afterwards, we use the magic values in Table 3 to partition the 4 data sets and compute the compression ratios. The results are shown in Fig. 4 to Fig. 8.

Table 3 Magic value of 4 data sets based on different S_{ave}

S_{ave}	magic value			
	data set 1	data set 2	data set 3	data set 4
256	0xB7	0xC1	0xFB	0xB7
512	0x137	0xB7	0x15A	0x1C7
1024	0x337	0x2FE	0x32B	0x317
2048	0xF7	0x7F5	0x51B	0x76F
4096	0x5BF	0x544	0x242	0xF6F

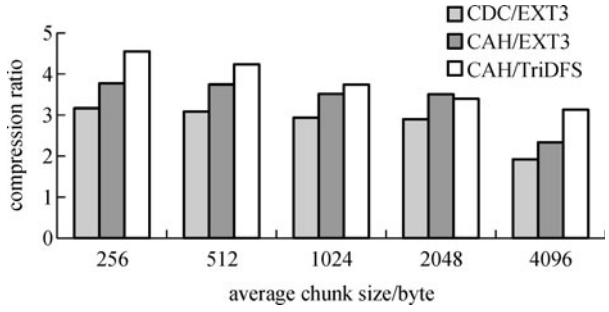


Fig. 4 Compression ratio of data set 1

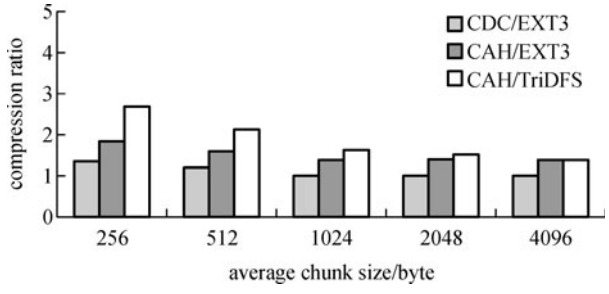


Fig. 5 Compression ratio of data set 2

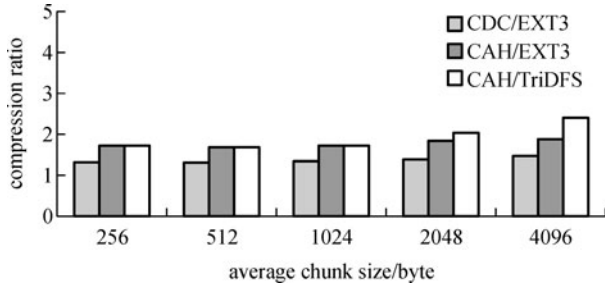


Fig. 6 Compression ratio of data set 3

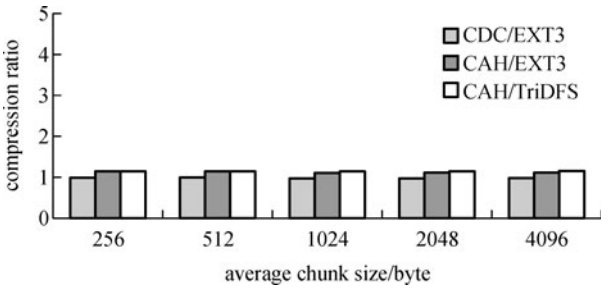


Fig. 7 Compression ratio of data set 4

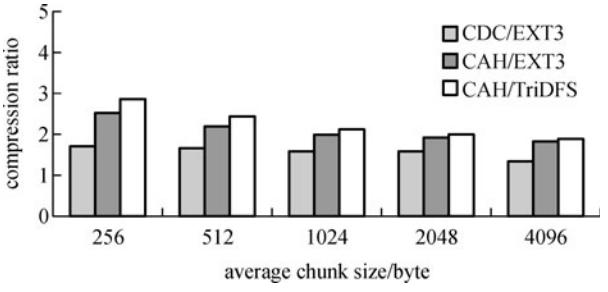


Fig. 8 Compression ratio of all 4 data sets

We can draw a few of conclusions from the results: a) CAH can effectively improve the data compression ratio. However, the improvements to each data set are different. The improvements to data sets 1, 2, 3 are more obvious than that to data set 4. This is because the byte value of data set 4 conforms to the assumption of uniform distribution of [0,255], while CAH can only improve the compression ratio of data sets that do not conform to that assumption; b) In most cases, TriDFS can further increase space compression ratio, but this is not the case for all. For example in data set 2, TriDFS instead decreases the compression ratio a little. This is because that with the growth of S_{ave}, if it exceeds the size of most files, then the unfilled space in the container increases. Therefore, larger S_{ave} may cause a negative impact on compression ratio; c) When adding up the individual improvements on each data set, the result is almost equal to the compression ratio improvement on the compound 4 data sets, which indicates that the redundancy among different file types is far smaller than that among files of the same type; d) On average, CAH can improve data compression ratio by 12.8%, and when together with TriDFS, the total compression ratio can be improved by 16.3%.

4.3 Throughput

In practice, besides compression ratio, throughput is also a key factor used to choose a compression technique.

In the evaluation, we blend the 4 data sets into one

compound data set, and measure the write/read throughput of the application. The write throughput is measured by writing the whole data set to the target machine and computing the amount of data that can be processed per second. The operations involved in the writing process includes computing fingerprints, looking up hash table, transferring chunks to the target machine, allocating container and offset in the container, and writing chunks to the disk. The read throughput is measured by opening each file in the data set, traversing and reading the whole content of it, then closing it. The operations involved in the reading process include chunk addressing, reading chunks from disk and transferring chunks to the initiator machine. The results of write and read throughput are shown in Figs. 9 and 10.

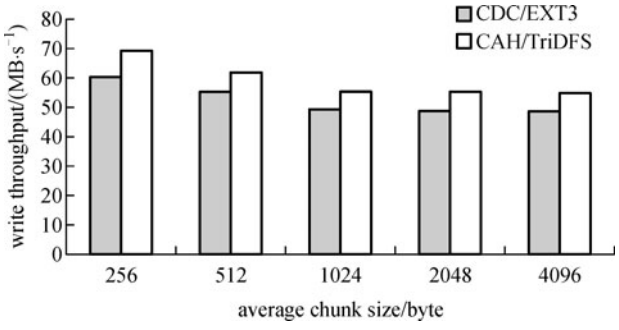


Fig. 9 Write throughput

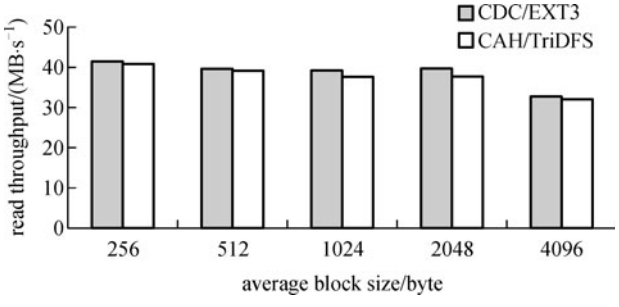


Fig. 10 Read throughput

We can draw a few of conclusions from the results: a) The write throughput is improved on average by 13.7%. This is because more duplicated chunks are identified by CAH and these chunks need not be transferred. Therefore, it is obvious that there is a direct relationship between compression ratio and write throughput; b) The read throughput decreases on average by 2.5%. This is because EXT3 store large files in sequential mode, while TriDFS scatters chunks into different containers and there are two more levels of indirection (from hash key to container and then to the chunk) in TriDFS so the average time in chunk addressing is increased. However, the decrease can be acceptable.

5 Conclusion and future work

This paper proposes CAH to represent the unique content characteristic of different file types and a corresponding algorithm of choosing anchors for CDC based on CAH. It has been proven that by exploiting the redundancy among files of the same type, the data compression ratio can be improved to some degree, while the performance lost of not identifying the redundancy between different file types is negligible. Furthermore, a few key parameters of CDC are tuned based on the date layout of underlying file system TriDFS, which improve the matching degree between the average chunk size and physical block size, thus improving the space compression ratio. The evaluation shows that CAH and TriDFS can improve on average the data compression ratio by 16.3%. and the write throughput by 13.7%. However, due to the overheads introduced by extra computing, the read throughput decreases on average by 2.5%, yet we think that this disadvantage is acceptable.

Future work includes: a) Converting the sequential version of CDC to the concurrent version, and move most computation to the graphic processing unit (GPU), which we expect will reduce the time spent on intensive computations such as computing the fingerprints; b) Improving the reliability of de-duplication storage systems. While de-duplication improves the storage utilization, it weakens the reliability of data. Because more than one file share common chunks, should a shared chunk be destroyed, all files that contain this chunk are destroyed also. Therefore, we should provide disaster toleration for de-duplicate storage systems.

Acknowledgements This work was supported by the National Natural Science Foundation of China (Grant No. 60673001).

References

1. Tony A, Biggar H. Data De-Duplication and Disk-to-Disk Backup Systems: Technical and Business Considerations. The Enterprise Strategy Group Technical Report. 2007
2. Biggar H. Experiencing in Data De-Duplication: Improving Efficiency and Reducing Capacity Requirements. The Enterprise Strategy Group Technical Report. 2007
3. Lillibridge M, Eshghi K, Bhagwat D, Deolalikar V, Trezise G, Camble P. Sparse indexing: large scale, inline deduplication using sampling and locality. In: Proceedings of the 7th USERNIX Conference on File and Storage Technologies. 2009
4. Cox L P, Murray C D, Noble B D. Pastiche: making backup cheap and easy. In: Proceedings of the 5th Symposium on Operating Systems Design and Implementation. 2002, 285–298
5. Quinlan S, Dorward S. Venti: a new approach to archival storage. In: Proceedings of the Conference on File and Storage Technologies. 2002, 89–101
6. Jain N, Dahlia M, Tewari R. TAPER: tiered approach for eliminating redundancy in replica synchronization. In: Proceedings of the 4th

USENIX Conference on File and Storage Technologies. 2005, 4: 21

7. Bobbarjung D R, Jagannathan S, Dubnicki C. Improving duplicate elimination in storage systems. ACM Transactions on Storage, 2006, 2(4): 424–448
8. Zhu B, Kai L, Patterson H. Avoiding the disk bottleneck in the data domain deduplication file system. In: Proceedings of the 6th USENIX Conference on File and Storage Technologies. 2008, 18
9. You L L, Karamanolis C. Evaluation of efficient archival storage techniques. In: Proceedings of the 21st IEEE Symposium on Mass Storage Systems and Technologies. 2004, 227–232
10. Manber U. Finding similar files in a large file system. In: Proceedings of the USENIX Winter 1994 Technical Conference. 1994, 1–10
11. Rabin M O. Fingerprinting by Random Polynomials. Center for Research in Computing Technology. Harvard University Technical Report TR-15-81. 1981
12. Brin S, Davis J, Garcia-Molina H. Copy detection mechanisms for digital documents. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. 1995, 398–409