Computing and Informatics, Vol. 35, 2016, 586–614

DATA DE-DUPLICATION WITH ADAPTIVE CHUNKING AND ACCELERATED MODIFICATION IDENTIFYING

Xingjun Zhang, Guofeng Zhu

Department of Computer Science and Technology
Xi'an Jiaotong University, Xi'an, 710049, China
e-mail: xjzhang@mail.xjtu.edu.cn, startzgf168@126.com

Endong Wang

Inspur(Beijing) Electronic Information Industry Co. Ltd.
100085, Beijing, China
e-mail: wed@inspur.com

Scott Fowler

Department of Science and Technology
Linköping University, Campus Norrköping, SE-601 74, Sweden
e-mail: scott.fowler@liu.se

Xiaoshe Dong

Department of Computer Science and Technology
Xi'an Jiaotong University, Xi'an, 710049, China
e-mail: xsdong@mail.xjtu.edu.cn

Abstract. The data de-duplication system not only pursues the high de-duplication rate, which refers to the aggregate reduction in storage requirements gained from de-duplication, but also the de-duplication speed. To solve the problem of random parameter-setting brought by *Content Defined Chunking* (CDC), a self-adaptive data chunking algorithm is proposed. The algorithm improves the de-duplication rate by conducting pre-processing de-duplication to the samples of the classified files

and then selecting the appropriate algorithm parameters. Meanwhile, FastCDC, a kind of content-based fast data chunking algorithm, is adopted to solve the problem of low de-duplication speed of CDC. By introducing de-duplication factor and acceleration factor, FastCDC can significantly boost de-duplication speed while not sacrificing the de-duplication rate through adjusting these two parameters. The experimental results demonstrate that our proposed method can improve the de-duplication rate by about 5 %, while FastCDC can obtain the increase of de-duplication speed by 50 % to 200 % only at the expense of less than 3 % de-duplication rate loss.

Keywords: Data de-duplication, self-adaptive, FastCDC

1 INTRODUCTION

The big data [1, 2] have the following features: volume, variety, value and velocity. The dataset is so huge that it is impossible to acquire, monitor, manage or handle within tolerable time by adopting common software tool. Therefore, searching for the way to manage and store these huge data is a problem that needs to be addressed.

The common solutions mainly include distributed storage (Hadoop), NoSQL data base (Cassandra), data compression technique and the latest data de-dup technology, in which all these methods used together for distributed storage and management in order to reduce the data volume. The current popular way to achieve distributed storage and management is to apply the MapReduce [3]. MapReduce breaks the task down into many tasks, all of which are tackled by several processors, and all the processing results are eventually integrated into a final result. The most distinctive feature of MapReduce is that it can improve the parallel processing ability of the system so as to handle big data more effectively. In terms of reducing data volume, it is chiefly achieved by data compression technique and data de-dup technology through eliminating the redundant data. Data de-dup technology generally includes file-oriented technique and chunk-oriented technique. The former, also called single instance storage [4, 5, 6], checks whether the files are identical according to their metadata and then stores the distinct ones. The later, including *Fixed-Sized Partitioning* (FSP) [7], CDC [7, 8, 9, 10, 11], SW [12] and FingerDiff-based technique [7], breaks the file into chunks, computes the hash of each chunk and detects whether the hash exists in the hash table to judge whether the chunk data are reduplicate and then stores the single chunk.

In the environment with a large amount of redundant data (i.e. E-mail system, backup application and data migration), data de-dup technology is capable of extremely saving storage space and network bandwidth. However, a huge amount of data results in a rapidly increasing need of storage space and the problem to manage big data. Based on our previous work [14], in this paper, firstly, a new backup system architecture based on data de-dup technology is proposed on the basis of MapReduce

mechanism, to meet the needs that backup application imposes on storage system in the big data era; secondly, two novel de-dup algorithms are put forward to quickly identify redundant data and effectively eliminate redundance; thirdly, a reliability mechanism of data and a scalability mechanism of system are presented to ensure system reliability and scalability; lastly, according to the specific dataset, test and analysis are conducted to the two novel de-dup algorithms.

2 BACKUP WITH DATA DE-DUPLICATION

2.1 System Architecture

Traditional backup system architecture based on data de-dup technology [13, 14, 15, 16, 19], mostly adopts C/S type, with the client generating backup data and forming backup task, while the server is storing and managing the data. The de-duplication methods during backup process can be classified into two types: one is on-line de-dup, operating during the data storage process which saves network bandwidth and storage space but occupies more CPU and spends more time on backup window. The other is post-processing de-dup, adopting some leisure de-dup after storing which does not occupy much CPU and spends less time on backup window but needs extra storage space, resulting in greater usage of the network bandwidth. However, no matter which method is adopted, backup data is generally stored and managed collectively in the traditional system architecture. As the data volume grows, the traditional system architecture cannot adapt to the big data application environment from the perspectives of either system performance or scalability.

Therefore, based on the traditional de-dup system architecture this paper presents a new one by reference to MapReduce, employing the methods of distributed storage and management. Figure 1 shows this system is composed of a backup client and server as well as several storage nodes. The backup client is mainly responsible for customizing backup tasks, recording their information, conducting backup, breaking files down into chunks, computing chunks' hash, etc. The backup server is in charge of managing metadata and storage nodes. The metadata includes the records of backup task ID, backup client host name, backup execution time, the number of files and backup data size and de-dup file metadata containing de-dup file ID, its number of chunks, and the hash sequence of each file. Storage node management, responsible for the automatic distribution of storage nodes of de-dup data, registers all storage nodes' information and maintains a global hash routing table. Storage nodes are responsible for de-dup detection and storage of de-dup data. Each storage node maintains a local hash table for keeping a record of the information of the corresponding chunk metadata (mainly including the hash, location, chunk length, number of chunk citations, etc.). The local hash table is used for querying and conducting the de-duplication detection to the hash value sent by the backup client.

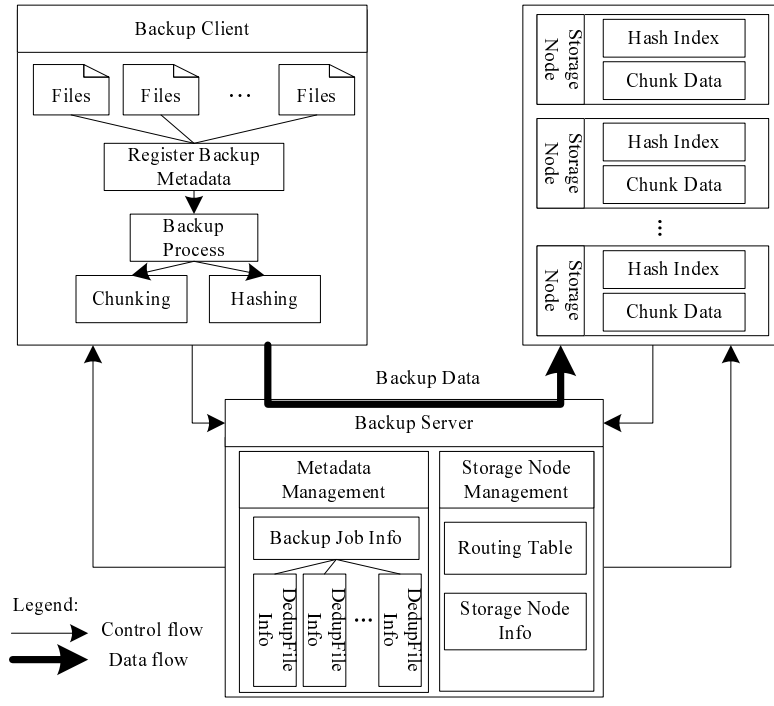


Figure 1. System architecture

2.2 Data Organization

The metadata of backup tasks, de-dup files and chunks should be stored to realize the functions of backup, plus do a restoration based on data de-dup in backup de-dup system. The first type is stored both in backup client and server; the second is stored in server and and the last is only stored in storage node. As Figure 2 shows, the backup task information mainly includes the the backup task's ID *Tname*, the creation time of backup task *Btime*, the backup client hostname *Bclienthost*, the backup source files *Bsrcpath* and the number of files contained by current task *Bfilenums*. De-dup file information records the metadata of de-dup file corresponding to the relevant backup tasks including the number of chunks broken down from the de-dup file, the chunk length and hash value. This information is stored by the Berkeley database belonging to the key-value db with "key" referring to the de-dup file ID *BackupJobNum* and "value" the de-dup file metadata. Chunk metadata, stored in hash table, is scattered in a number of storage nodes, each of which maintains a local hash table. Each table records the basic metadata of chunk corresponding to the relevant local storage node and this metadata mainly includes hash (Hash1, Hash2, etc.), chunk location *Loc*(C), chunk length *Len* and number of citations *References*.

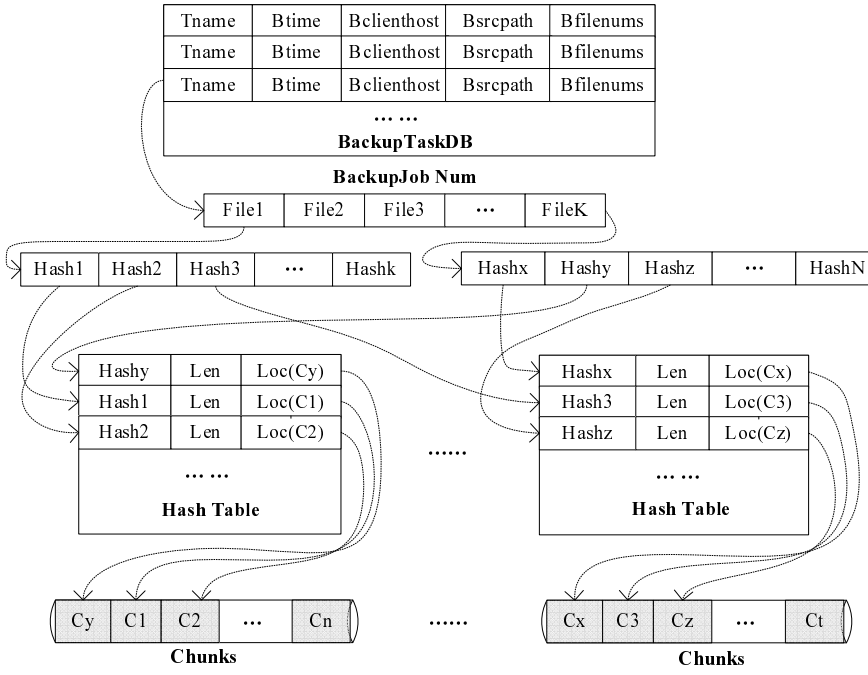


Figure 2. Data organization

The hash table adopts the management of the secondary index structure to detect quickly whether the hash is existent or not.

2.3 Process of Backup and Restoration

Since the de-dup of online source end is used, the backup data should be partitioned and the hash of all the chunks in the backup client computed. Next the backup server conducts the management of the backup and de-dup metadata. After completing these tasks, the storage node carries out the de-dup detection. The backup client only transmits chunk data whose hash value does not exist, which, therefore, can save network bandwidth and storage space simultaneously. Next the specific process of backup and restoration will be introduced in detail.

2.3.1 Backup

Figure 3 shows there are nine steps in the whole de-dup process: firstly, backup client customizes a backup task, stores the record, and then sends it to backup server; secondly, the server registers backup task and returns the result; thirdly, the client starts to backup by breaking down file into chunks and computing hash and then sends de-dup file metadata *FileInfo* as well as the corresponding hash

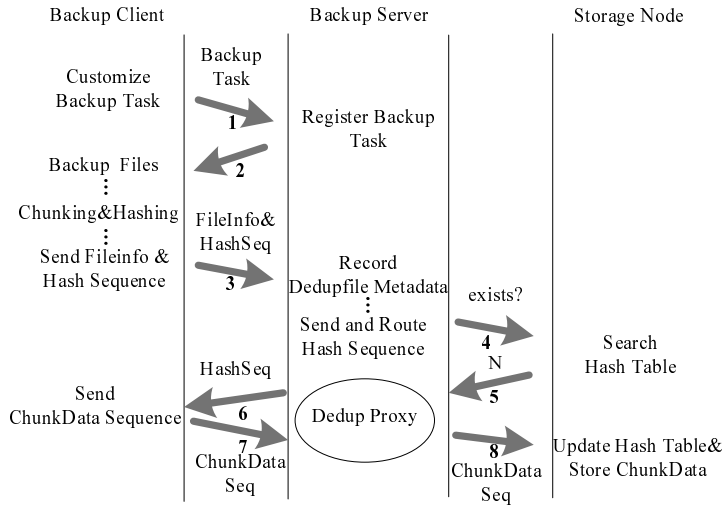


Figure 3. Backup process

sequence *HashSeq* to the server; fourthly, the server records de-dup file metadata and then based on the hash routing table sends the corresponding hash sequence to the relevant storage node; meanwhile, after having received the hash sequence, the storage node conducts the de-dup detection to it by searching hash table and returns the detection result to the server; afterwards, on the basis of the returned result, the server forms a new sequence for hashes (Hash Seq) that do not exist and return them to the client; after that, based on the received hash sequence, the client sends the corresponding chunk data *ChunkData Seq* to the server; then the de-dup proxy inquires the hash routing table and sends the chunk to the corresponding storage node; lastly, the storage node updates the local hash table and stores the chunk data.

2.3.2 Restoration

The restoration process is the inverse process of the backup. The former can be obtained based on the latter: the client user selects the particular backup task required to be restored and then sends the request to the server. The server inquires the de-dup file information based on the records of the backup tasks and then attains the corresponding chunk information and hash sequence by referring to the de-dup file information. After that, on the basis of the hash routing table, the chunk is read from the corresponding storage node, which is returned to the client.

3 OPTIMIZATION WITH ADAPTIVE CHUNKING AND FAST MODIFICATION IDENTIFYING

The data de-dup technology [18] is thought to include the technique of the identical and similar data detection and coding. Since the identical data detection is the research focus of this paper, the similar data detection and the coding technique will not be discussed.

The identical data detection technique can be classified into three types: the file-oriented de-dup, the chunk-oriented FSP, variable length de-dup, and the byte-oriented de-dup. The first type conducts de-dup to the whole file by comparing the metadata of files and only stores the single instance from many identical files which will be automatically mapped into the single instance by pointer. The merit of this de-dup type lies in its high de-dup speed, which is suitable in the application environment with large number of small and repeated files, such as file system and e-mail system. The third type, that is, the byte-oriented de-dup, with its high de-dup rate but low speed, conducts de-dup by comparing the bytes. Particularly, as the data volume grows, its scalability and applicability become terrible. The type can possess not only satisfactory de-dup rate but also high speed. Therefore, this de-dup type is widely employed in various systems.

The chunk-oriented de-dup [7] includes FSP, CDC, FingerDiff-based variable length and Sliding-window Chunking.

FSP [7, 18] breaks file stream down into chunks with a fixed size, computes the chunk hash and then by searching the hash table for the de-dup detection; only the chunks whose hashes don't exist are stored by the storage node.

CDC [7, 18] works with a fixed-size window sliding on the file stream by byte and the weak hash in the window continuously computed; meanwhile, the weak hash is detected as to whether meets the predefined conditions; if yes, record the offset in the file stream and set a breakpoint for it; if not, continue the window sliding. When the two breakpoints are found, take the data between the two points to form a chunk, compute its strong hash, conduct the de-dup detection to it and then only store the chunks whose hashes don't exist as well as update the hash table.

FingerDiff-based de-dup [7, 18] is an improvement on the basis of CDC for solving the metadata expansion. The backup file stream is firstly partitioned by CDC and then the chunks are combined into a big chunk according to the preset number of chunks; after that, the chunk hash is computed and then experiences the de-dup detection to see whether the chunks are reduplicate or not; if reduplicate, delete the reduplicate chunks and if not, break down the big chunk into smaller ones until the smallest single chunk is found and then store it with the rest remain combined.

SW de-dup technology [7, 18] incorporates the merits of FSP and CDC. The main idea is similar to that of CDC, that is to say, a fixed-size window slides on the file stream by byte and with every single sliding, for data in the window its checksum is computed by rsync and then the first matching detection is conducted; if it is matching, adopt the strict SHA-1 Hash function to compute its fingerprint

and then another de-dup detection is conducted and if it is reduplicate this time, after recording it, the sliding window skips the reduplicate chunks and moves on while at the same time the partitioned chunks and the fragment chunks arising before reduplicate chunks should be recorded and stored; if it is not matching, then the window continues sliding and when it is still mismatching after a window-length sliding, computes the checksum and hash value of the data in the window and then stores the chunk.

The advantage of FSP is high de-dup speed, while the disadvantage is its sensitivity to the modified data (in other words, it cannot effectively detect the redundant data once the data are modified), hence its low de-dup rate. Meanwhile, a large number of metadata will be generated if the data are broken down into smaller chunks. However, CDC can effectively find the redundant data from the modified data and so can get the high de-dup rate, but its disadvantage is low de-dup speed due to its computations of hash frequently in the process of sliding window and it may also generate a large quantity of metadata. The advantage of FingerDiff-based technique is high de-dup rate and it also decreases the metadata dramatically; however, it is too complex to accomplish besides its low de-dup speed. The advantage of sliding-window chunking is high de-dup rate because it breaks down files into fixed size which can be easily managed, so it is suitable for fine-grained matching [9]. However, it is too complex with low de-dup speed and tends to generate fragments during the de-dup process.

CDC is the most widely used algorithm in the de-dup system. The TTTD algorithm introduced in [30] is an improvement and optimization over CDC algorithm. It restricts the range of chunk length by setting two thresholds and increases the probability of boundary point matching by setting two divisors, both of which aim at increasing the de-dup rate and decreasing the network transmission consumption. However, corresponding algorithm is adopted according to different environment; for example, TTTD is adopted to obtain higher de-dup rate in the environment with large number of modified files and while SIS or FSP is adopted to get faster de-dup speed in the environment with large number of identical files but few modified files. In order to have the TTTD algorithm further improved from the perspectives of de-dup speed and rate, a novel self-adaptive CDC algorithm to improve the de-dup rate and a novel FastCDC algorithm to accelerate the de-dup speed based on the TTTD is proposed.

3.1 Adaptive Chunking

There is a preset condition in CDC [30] used to define the chunk breakpoint of file. Given a sequence $S = s_1, s_2, \ldots, s_n$, with h representing a fingerprint function and l a window length, there is a D-match at k if, for some pre-determined $r < D$,

$$h(W) \bmod D = r \tag{1}$$

where $W = s_{k-l+1}, s_{k-l+2}, \ldots, s_k$ is the subsequence of length l preceding the position k in S. All chunk boundaries that meet the above condition will be found when the window slides on the S sequence by byte. The value of residue r plays no role in our analysis and we use the value $D-1$ in our algorithm for consistency. However, the value of D determines the chunk length indirectly. The bigger D is, the lower the matching probability is and the longer the average chunk will be; and when the above conditions are opposite, the results are opposite too. Meanwhile, the de-dup rate is determined by the matching situation in condition 1. It is not true that the smaller or bigger D is better. We must find an appropriate value of D in correspondence with different file types and content to gain a high de-dup rate. However, it is still a challenge to find the proper value of D with the uncertain data in the backup process.

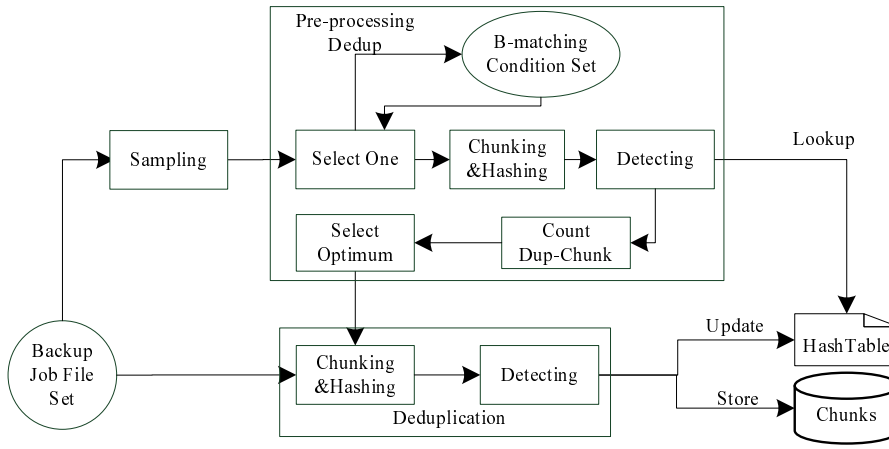


Figure 4. Self-adaptive CDC

Based on the above analysis, a novel self-adaptive CDC algorithm is proposed to select the proper value of D to increase the de-dup rate. As Figure 4 shows, firstly, the backup job file set is classified by file type and then stratified sampling according to file length is conducted among files with the same type. After that, the sampling file set which is used to determine the value of D through pre-de-duplication is formed. While doing the pre-de-duplication, the real de-dup is not conducted, since only the number of redundant chunks and the total number of chunks is recorded to compute the pre-de-duplication rate. The specific process of pre-de-duplication is shown as follows: firstly, the B-matching condition set which is the collection of D on different file types is preset based on a lot of practical process; secondly, the B-matching conditions are selected successively to conduct the chunking and computing of hash on sampling file set; thirdly, the de-dup detection is carried out and the numbers of dup-chunks and total chunks are recorded respectively; finally, the formula, pre-de-duplication rate = (the number of dup-chunks)/(total number

of chunks), is computed and the optimum D whose corresponding pre-de-duplication rate is the highest is selected as the final breakpoint condition value.

The de-dup rate is improved through the pre-processing de-dup which self-adaptively adjusts the parameter of the algorithm to adapt to the unpredictable data. The effectiveness of the algorithm will be verified by experiment in the last section of this paper.

3.2 Accelerated Modification Identifying

The appearance of the modified files is not usual during the backup process whether it is full backup, incremental backup or differential backup. Under this circumstance, if CDC, SW or FingerDiff-based techniques are still used for de-dup detection to the whole files, the de-dup process is not only inefficient but also clumsy. Meanwhile, even if a file is modified, it is only part of it, and how to rapidly detect the modified part is the key to increase the algorithm performance. Therefore, the novel de-dup algorithm, FastCDC, is proposed to solve the above problem.

Figure 5 shows the main idea of FastCDC:

1. the file stream is broken down into variable-length chunks by CDC and the position of last breakpoint (lbp) after blocking is recorded simultaneously;
2. a fixed-size chunk is partitioned from the lbp and the fixed-size position is recorded at the same time;
3. starting from the last fp, sliding window continues to conduct the file-breaking by CDC;
4. according to this procedure, every variable-length chunk is followed by a fixed-size chunk and if length of the final file is shorter than the fixed length then the rest is integrated into a chunk and the whole process is over.

This algorithm is capable of rapidly identifying the modified part in the file and guaranteeing fast de-dup detection to all the files. Altogether there are three locations where the parts of files are modified: in the variable-length chunks, fixed-size chunks or the breakpoint window. If the data in the breakpoint window have not been modified, then the modified data are either in variable-length chunks or in fixed-size chunks. The first situation only affects the variable-length chunk in question while the second situation affects both the fixed-size chunk in question and the later variable-length chunks; if the data in breakpoint window have been modified, by controlling the size of the chunks, at most three chunks are affected. However, in view of the probability, data in the breakpoint only occupy quite a small part of the whole file, so the average probability for its modifying is quite small. The key to this algorithm is how to set fixed-size, the acceleration factor. Generally speaking, if the fixed size is bigger, the de-dup speed will be higher; on the contrary, if the fixed-size is too small, though the de-dup rate is higher, the de-dup speed will be decreased. Actually, in the real test, high de-dup speed will absolutely result in low

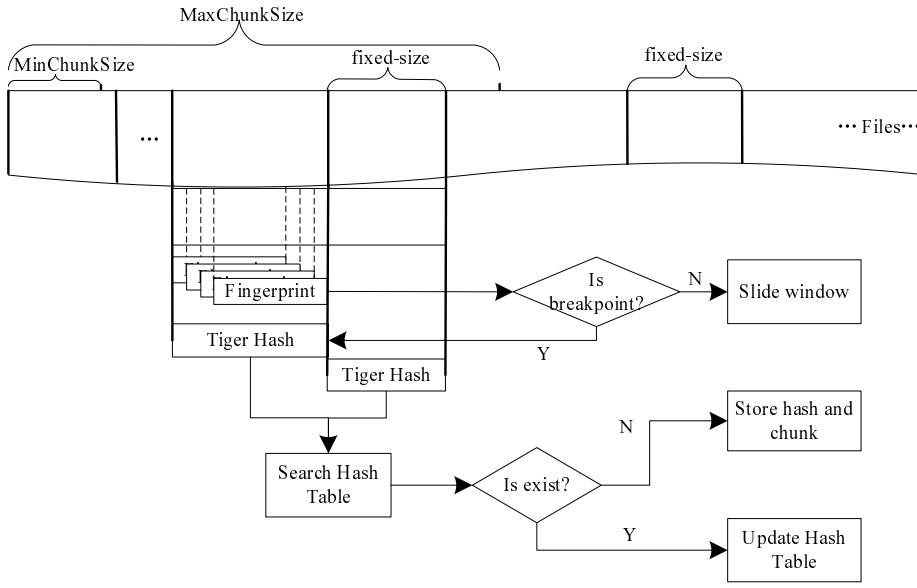


Figure 5. FastCDC

de-dup rate, which, however, is contradiction to the original intention of the data de-dup system. Therefore, further optimization is needed to alleviate this problem.

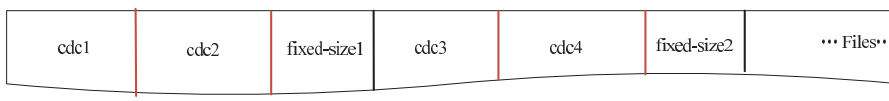


Figure 6. Optimization of FastCDC

The main idea of the above algorithm is to break file down into chunks through the CDC breakpoint. The chunk before the breakpoint is a variable-length chunk while the one after the breakpoint is fixed-size chunk. It is found that data modification in fixed-size chunk will affect the later variable-length chunk. Since the number of the fixed-size chunks is larger, the probability for its modification is larger and then there are more chunks that will be affected, the result of which is that the amount of detected redundant data is smaller. Therefore, in order to improve the de-dup rate and reduce the weight occupied by fixed-size chunks, the variable-length chunks can be increased. Figure 6 shows, firstly partition two variable-length chunks successively and then partition a fixed-size chunk (cdc1 and cdc2 are variable-length chunks and cdc2 is one that is identified and partitioned completely based on its content, fixed-size1 is a fixed-size chunk). Since the number of successive variable-length chunks determines the de-dup rate, we call this number de-dup factor. High de-dup

speed and rate are possible by setting a proper de-dup factor. The next experiment part will test and analyze this algorithm and the traditional CDC algorithm.

3.3 FastCDC Theoretical Analysis

There are several important parameters in FastCDC, including sliding interval [*MinChunkSize*, *MaxChunkSize*] (*MinChunkSize* indicates the minimum chunk length, and the *MaxChunkSize* represents the maximum chunk length), de-dup factor *VChunksNum* which is the number of continuous variable-length chunks and fixed-size acceleration factor which represents length of fixed-size chunk. The chunk breakpoint condition is defined as Equation (1).

For the randomness of RabinHash, the distribution of chunk breakpoint will be theoretically random. Therefore, the average expected chunk length of CDC is:

$$E\_Len\_Var = D + MinChunkSize \tag{2}$$

Considering the fixed-size chunk, the average expected chunk length of FastCDC is:

$$E\_Len = \frac{VChunksNum}{VChunksNum + 1} * E\_Len\_Var + \frac{1}{VChunksNum + 1} * \textit{fixed-size} \tag{3}$$

According to the above analysis, the average number of sliding window based on the CDC algorithm is:

$$P_{CDC} = \frac{Length}{E\_Len\_Var} * D \tag{4}$$

The average number of sliding window using FastCDC algorithm is:

$$P_{FastCDC} = \frac{Length}{E\_Len} * \frac{VChunksNum}{VChunksNum + 1} * D \tag{5}$$

Compared with CDC, the performance of FastCDC is improved by:

$$1 - \frac{P_{FastCDC}}{P_{CDC}} = \frac{\textit{fixed-size}}{VChunksNum * E\_Len\_Var} \tag{6}$$

According to the Equation (2) and (6), there are four main variables influencing the performance of FastCDC, including *fixed-size*, *VChunksNum*, *D* and *MinChunkSize*. Fixed-Size, which is acceleration factor, influences the de-dup speed of FastCDC, and the bigger the value of acceleration factor is, the faster the de-dup speed of FastCDC will be. *VChunksNum*, namely de-dup factor, determines the de-dup rate of FastCDC and the bigger the value of de-dup factor is, the higher the de-dup rate of FastCDC will be. Based on Equation (6), the conclusion can be drawn as follows: with the *VChunksNum*, *D* and *MinChunkSize* unchanged, the bigger the acceleration factor is, the faster the de-dup speed of FastCDC will be.

The algorithm will turn into CDC when the de-dup factor is set to infinity, and it will become FSP if the acceleration factor is set big enough or the de-dup factor is set to zero. Therefore, we can adjust the performance of FastCDC algorithm to meet different requirements by setting different de-dup factor and acceleration factor.

4 SYSTEM SCALABILITY ANALYSIS

Compared with the traditional system architecture, the proposed architecture divides the backup task with enormous data into smaller ones and then assigns these smaller tasks to many storage nodes through a set of mapping mechanism. The backup server is indeed a de-dup agent and the actual de-dup detection and chunk storage processes are conducted at every storage node. In this way, the capability of the server can be lessened and also the parallel processing capability of the system can be improved. However, with the data volume increasing, the server's capability and the storage nodes' storage capacity will gradually turn into the bottleneck of the system. Therefore, it is extremely urgent to find the way of ensuring system scalability.

The two typical research findings on scalability of de-dup system are:

1. [27] presented a scheme taking advantage of the locality feature of the backup data stream. With the proposed method, the system is capable of conducting de-dup detection rapidly in the big data environment by introducing an index buffer and adopting Bloom filter technique.
2. [28] also proposed to exploit potential locality of data and use sampling techniques, that is, divide the backup data stream into relatively large chunks and then conduct de-dup detection according to the sparse index established by sampling in advance.

Since the sparse index is smaller compared to the whole index table, it can be placed in RAM, which, to a certain degree, solves the problem of memory capacity limitation and disk bottleneck. However, the above two solutions are all applied in central system and cannot be easily extended to the distributed system architecture. Therefore, a novel scalability mechanism is needed in distributed system. In the post-processing de-dup system proposed in [15], the detection and update of fingerprints are dispersed in a cluster. A two-phase de-dup detection technique is adopted to further improve the system scalability.

As presented in Figure 1, the backup client backs up data to the server and the actual data are stored in multiple storage nodes. Backup client mainly conducts the backup pre-process, that is, generates the backup task, segments files and computes hashes. The server is responsible for managing backup data as well as the other more important job which divides a complex de-dup detection task into multiple smaller ones and then assigns them to different storage nodes tactically. How to assign them is a key problem in assuring system scalability. Firstly, the information of all the storage nodes, including IP address, port number and so on, is recorded

in a hash routing table by backup server; secondly, a set of mapping rules about how to map hashes to storage nodes are built; after that, a new hash routing table should be created when a storage node is added and both the original storage nodes' information and the new one's are registered. Now, the backup server can conduct mapping based on the new hash routing table when a new de-dup task needs to be processed. If the restoration process is requested, the backup server will firstly find the corresponding hash routing table according to the create date of requested backup task and then read the data from the corresponding storage nodes. By using this mechanism, we can expand storage nodes dynamically, which improves not only the parallelism of the system but also the de-dup speed and thus adapts to the big data application environment.

5 SYSTEM RELIABILITY ANALYSIS

The implementation of data de-dup technique leads to a single instance shared by multiple files [7]. With large amount of redundancy eliminated, the loss of one datum or several key data chunks may have severe impact on all files involved, making them inaccessible. How to ensure the reliability of de-duplicated data is one of the key research problems for de-dup technique. To solve this problem, some researchers [7, 22, 23] introduced redundant replication technology into de-dup system, thus having improved system availability through replicating key data chunks tactically. Others [7, 24, 25] proposed the use of erasure coding technology, that is, increase the availability of de-dup system by adding redundant data. The two techniques mentioned above, both essentially introduce redundant data to improve the system's reliability, but with different technical characteristics and scope of application [7]. In our system, we proposed a scheme which improves system reliability without impairing de-dup rate. Key data chunks, for which duplicates are created and distributed to multiple storage nodes, are identified on the basis of reference number.

As presented in Figure 7, two types of table exist in our availability scheme. The one named Hash Routing Table is kept on the backup server, recording the matching relationship between hash values and storage nodes. The other kind of table exists in each storage node, called metadata index table, recording the metadata of each data chunk in a key-value database. The metadata mainly contain information like hash value, chunk length, storage location, reference number, etc. According to Figure 7, the reliability assurance scheme is realized as follows:

1. The backup client generates a sequence of hash values, and sends them to the backup server.
2. For each hash value received, the backup server queries hash routing table, gets the corresponding storage node to which the data chunk represented by the hash value is sent, and then triggers the storage node service process. In the hash table, HV represents the range of first byte of the hash values; SN indicates information about the storage nodes, such as IP address, port number, etc.

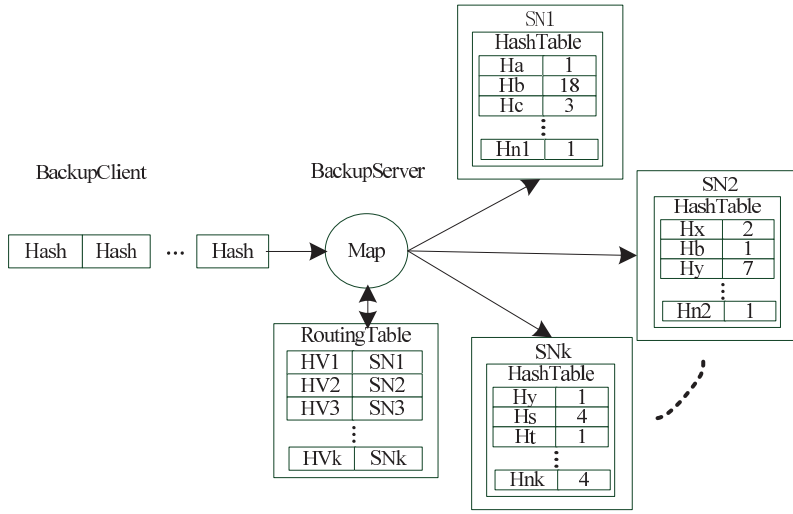


Figure 7. Reliability mechanism

3. Hashes received are sent to corresponding storage nodes according to hash routing map. For example, $(Ha, Hb, Hc, \ldots, Hn_1) \rightarrow SN1, (Hx, Hy, \ldots, Hn_2) \rightarrow SN2, \ldots, (Hs, Ht, \ldots, Hn_k) \rightarrow SN_k$. For a hash value that does not exist, the storage node should not only update metadata index table, but also store the corresponding chunk.
4. When the hash value received is detected as already existing, the duplicate hash value should also be mapped to a storage node with reference to hash routing table. The corresponding storage node will not store the data chunk again; instead it updates the reference number for this hash value in its metadata index table and sends the number to backup server.
5. When the backup server found the corresponding reference number of a hash value to be a multiple of a predetermined threshold (take 7 as an example), it will reselect a byte in the hash as the mapping condition. The reselecting formula is references/7. That is, when the reference number reaches 7, choose the second byte of the hash value as the re-mapping condition. Assuming H is the hash value to be considered, the first byte of its value is Hb, and then according to the mapping relationship listed in step c, the content of this hash value is stored in SN1. When the reference number reaches 7, the server selects its second byte as re-mapping condition. If its second byte is Hx, the server will send a copy of data chunk to SN2.
6. If the reference number of a hash value (still take H as an example) continues to increase and meet the condition of adding duplicates, the server will again calculate references/7 to determine which byte will be used. When the reference

number of H reaches 14, the server takes the third byte of H as the mapping condition and sends a duplicate to the corresponding storage node. In order to maintain de-dup rate at an appropriate level, we define that the number of duplicates of a single chunk should not exceed 4.

Using the availability assurance mechanism described above, when a data chunk on a storage node was lost or went wrong, the backup server would automatically find its copies according to the first four bytes of its hash value, and restored it with the right content. Compared with other redundant replication technology, the above scheme gets the following advantages:

1. There is no need to manage redundant copies with extra metadata, thus availability is assured with the least implication on de-dup rate.
2. It is flexible and easy to implement, and could restore key missing data chunks in an efficient way.

6 PERFORMANCE EVALUATIONS

6.1 Experimental Environment

We used 4 PCs in our testing environment, with one of them used as backup client and server simultaneously and the rest as storage nodes. The hardware and software environment of all PCs are roughly the same, with the details as follows: Windows XP, Pentium(R) Dual-Core CPU E5500 @ 2.80 GHz, RAM 4 GB (Sharetronic/Kingston DDR3 1 333 MHz), 500 GB/7 200 RPM and Cache 16 MB.

6.2 Dataset

We select four file types, including text file type, picture file type, audio file type and rar file type, to test the algorithms' performance. The selected files mainly come from daily data in our lab (project meetings, daily e-mail and papers, etc.). The pictures and music are from several websites (Baidu, Sogou and Google, etc.). The details are shown in Table 1. Doc and ppt type have a large amount of modified data, pdf and audio file type have a small quantity of modified data, while jpg and gif file type have almost no modified data.

The daily email data in testing dataset I were rearranged according to their corresponding date of month. Data of fifteen months from January 2012 to March 2013 are obtained; the data size is 3 GB, the number of files totals 1 132 and the file types include jpg, pdf, ppt, doc, zip, wmv, etc., the details are shown in Table 2.

File Type	Number	Size	Sources of Data
doc	1 960	1.87 GB	Project documents and e-mail attachments
pdf	1 915	2.42 GB	Downloaded papers from project team members
ppt	176	768 MB	Project documents and Baidu Library
jpg, gif	2 817	1.56 GB	Google and Baidu pictures
mp3, wma	271	1.12 GB	Baidu and Sogou music
rar	10	5.44 GB	Compressing files containing doc, ppt, jpg and mp3
total	7 149	13.2 GB	Daily data in lab and website

Table 1. Dataset I

Data Set	Number	Size/MB	Month	Data Set	Number	Size/MB	Month
1	123	224	2 012.1	9	677	1 730	2012.9
2	195	424	2 012.2	10	741	1 956	2012.10
3	269	593	2 012.3	11	773	2 130	2012.11
4	323	728	2 012.4	12	828	2 284	2012.12
5	390	939	2 012.5	13	929	2 570	2013.1
6	454	1 137	2 012.6	14	1 045	2 816	2013.2
7	532	1 341	2 012.7	15	1 132	3 072	2013.3
8	600	1 536	2 012.8				

Table 2. Dataset II

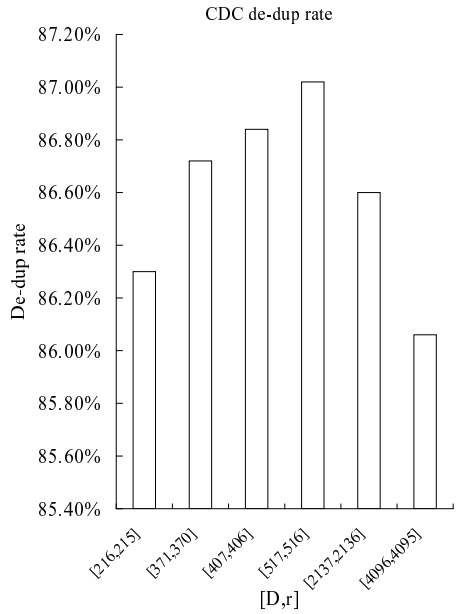
6.3 Experimental Results Analysis

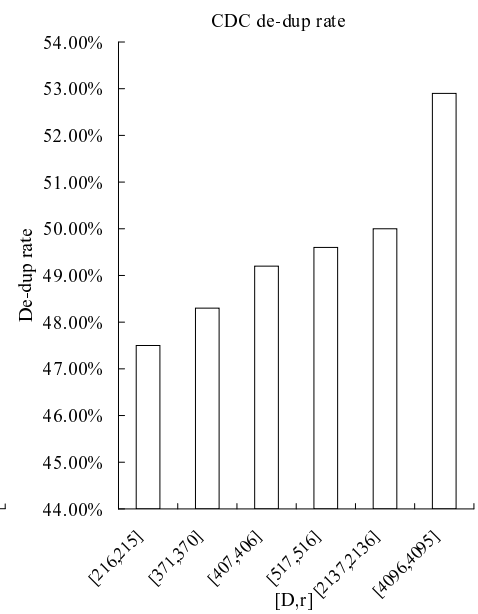
6.3.1 Self-Adaptive CDC Performance

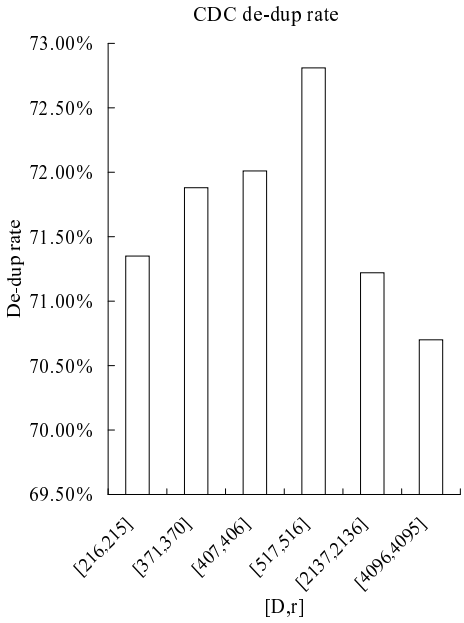
To verify the effectiveness of self-adaptive CDC, we select different Ds in the de-dup process to test the above dataset in the de-dup process. The r is set to $D - 1$ according to the previous assumption since it has no effect on the de-dup rate. The result is shown as follows.

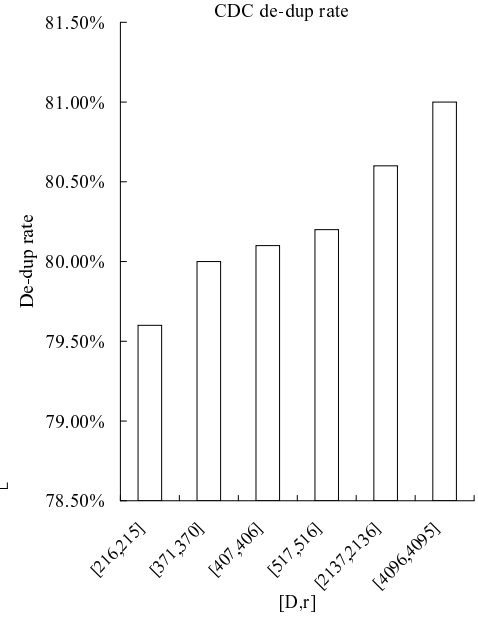
As shown in Figure 8, the X axis represents the values of D and r, and Y axis indicates the de-dup rate (de-dup rate), equaling to (source data size − de-duplicated data size)/source data size.

According to the test results, CDC will get different de-dup rates when we select different Ds for any file type. For ppt, doc and rar type with a large number of modified data, to obtain a higher de-dup rate, we should select a proper value for D, neither too smaller nor too bigger. For instance, both ppt and doc types can attain the highest de-dup rate when the D equals to 517. While for audio, pdf and pictures file types which have no modified data or the data of which are not easy to be modified, the D value mainly affects the chunk length, that is, the bigger D is, the longer chunk will be, and the smaller D is, the shorter chunk will be. However, the D value can not be set too large for it will decrease the probability of breakpoint condition matching and then increase the number of times to compute hash. Therefore, only by using the self-adaptive CDC can we set the appropriate D for different file types.


CDC de-dup rate
87.20%
87.00%
86.80%
86.60%
86.40%
86.20%
86.00%
85.80%
85.60%
85.40%
De-dup rate
[216,215]
[371,370]
[407,406]
[517,516]
[2137,2136]
[4096,4095]
[D,r]


CDC de-dup rate
54.00%
53.00%
52.00%
51.00%
50.00%
49.00%
48.00%
47.00%
46.00%
45.00%
44.00%
De-dup rate
[216,215]
[371,370]
[407,406]
[517,516]
[2137,2136]
[4096,4095]
[D,r]


CDC de-dup rate
73.00%
72.50%
72.00%
71.50%
71.00%
70.50%
70.00%
69.50%
De-dup rate
[216,215]
[371,370]
[407,406]
[517,516]
[2137,2136]
[4096,4095]
[D,r]


CDC de-dup rate
81.50%
81.00%
80.50%
80.00%
79.50%
79.00%
78.50%
De-dup rate
[216,215]
[371,370]
[407,406]
[517,516]
[2137,2136]
[4096,4095]
[D,r]

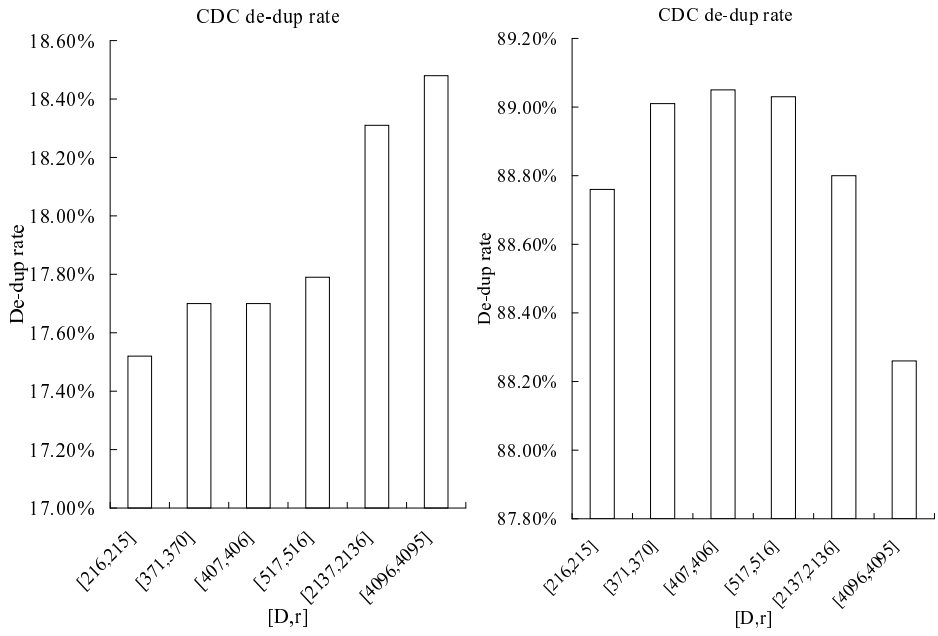


Figure 8. Performance evaluation on different file types

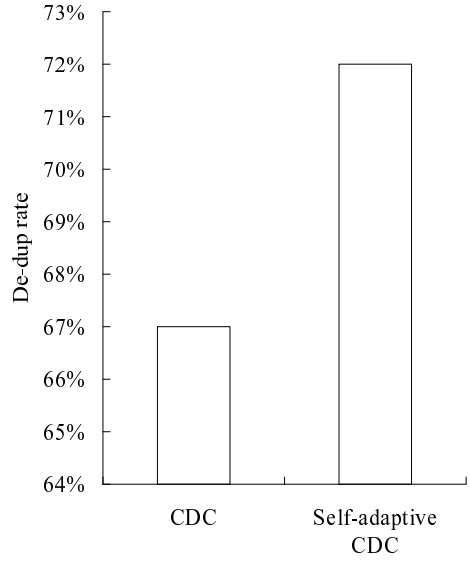


Figure 9. Testing on self-adaptive CDC and CDC

Next we use both the traditional CDC and the self-adaptive CDC respectively to test the whole dataset. The testing results are shown in Figure 9: X axis represents the two algorithms and Y axis indicates the de-dup rate. From the diagram, we can conclude that in comparison with traditional CDC, self-adaptive CDC can increase the de-dup rate by about 5 percent since D is determined dynamically according to different file types.

6.3.2 FastCDC Effectiveness

We use FastCDC and CDC to test the above dataset respectively as well as set different de-dup factors and acceleration factors. Testing results are shown as follows (Figures 10–15). In the figures, X axis indicates the de-dup factor, for which number 1 to 6 represents the number of continuous variable-length chunks. The horizontal line stands for the CDC algorithm. Y axis represents the de-dup rate and the de-dup speed respectively corresponding to the de-dup factors and acceleration factors of FastCDC. The de-dup rate is defined as above, and the de-dup speed equals to (source data size)/(backup window), the unit of which is MB/S.

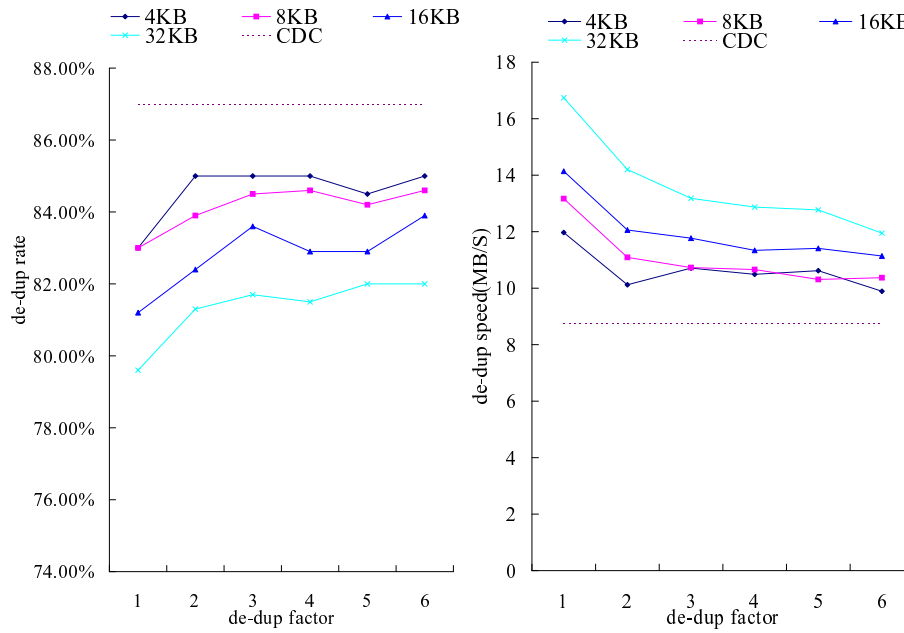


Figure 10. Testing on doc file type

As the testing result shows, for FastCDC algorithm, with the de-dup factor increasing, the de-dup rate will increase correspondingly, but the de-dup speed decreases; and with the acceleration factor increasing, the de-dup speed will increase correspondingly, but the de-dup rate decreases. Further analyzing the above test

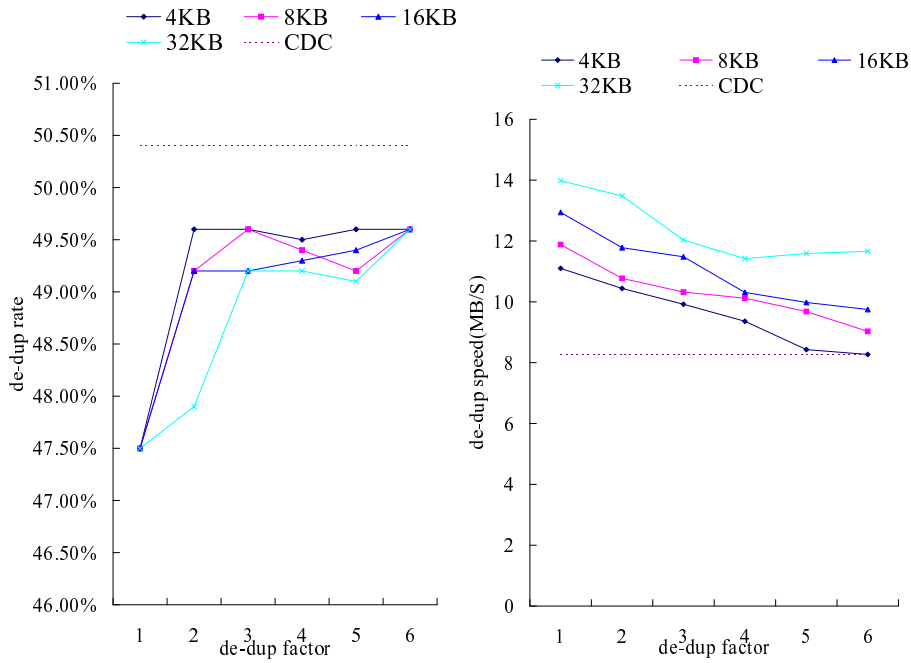


Figure 11. Testing on pdf file type

results, we conclude that all of the above 6 file types will attain an increase on de-dup speed ranging from 50 % to 200 % when the de-dup factor is 1, but the de-dup rate of them will be impaired to different degrees (the doc file type's de-dup rate decreases by 4 %–8 %, the pdf decreases by about 3 %, the ppt decreases by 15 %–29 %, the picture is seldom affected but increases a little, the audio decreases by 6 %–8 % and the rar decreases by 15 %–40 % which has the biggest impact). However, if the value of de-dup factors is not smaller than 2, the impairment on de-dup rate will be not greater than 5 % by FastCDC, but the de-dup rate will increase substantially ranging from 50 % to 200 %. Therefore, for the file types that have a larger number of modification data, for instance, doc and ppt, etc., the de-dup speed will increase significantly by setting larger de-dup factor and acceleration factor (the de-dup factor is set to 6 and the acceleration factor to 32 KB) under the premise of minimal impairment of de-dup rate; while for the file types with a small number of modification data, for instance, mp3 and pdf, etc., the de-dup speed will increase greatly at the expense of small impairment of de-dup rate by setting a small de-dup factor and a big acceleration factor (the de-dup factor is set to 3 and the acceleration factor to 32 KB). However, for the file type with no modified data, not only the de-dup rate can be increased but also the de-dup speed by setting the smallest de-dup factor and the largest acceleration factor.

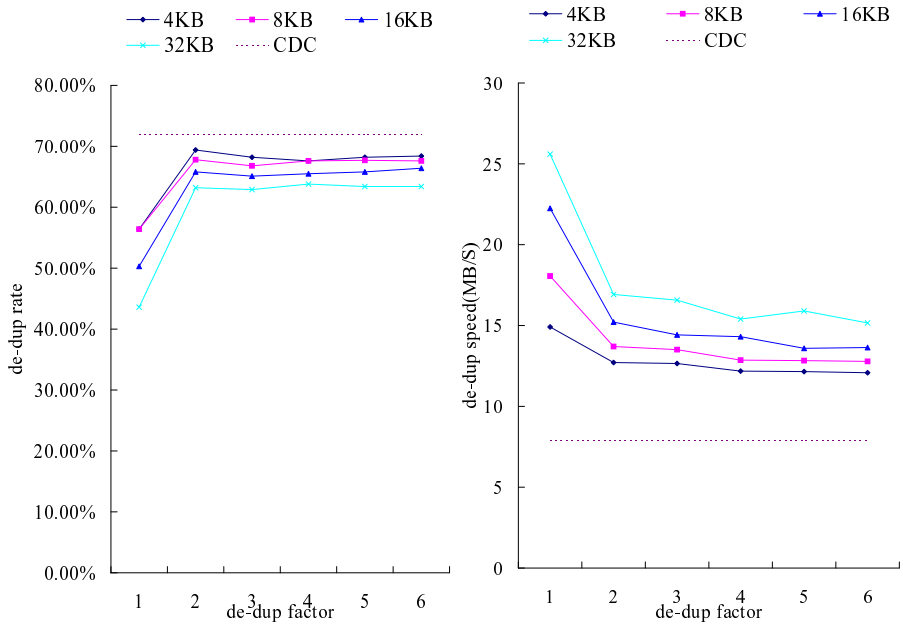


Figure 12. Testing on ppt file type

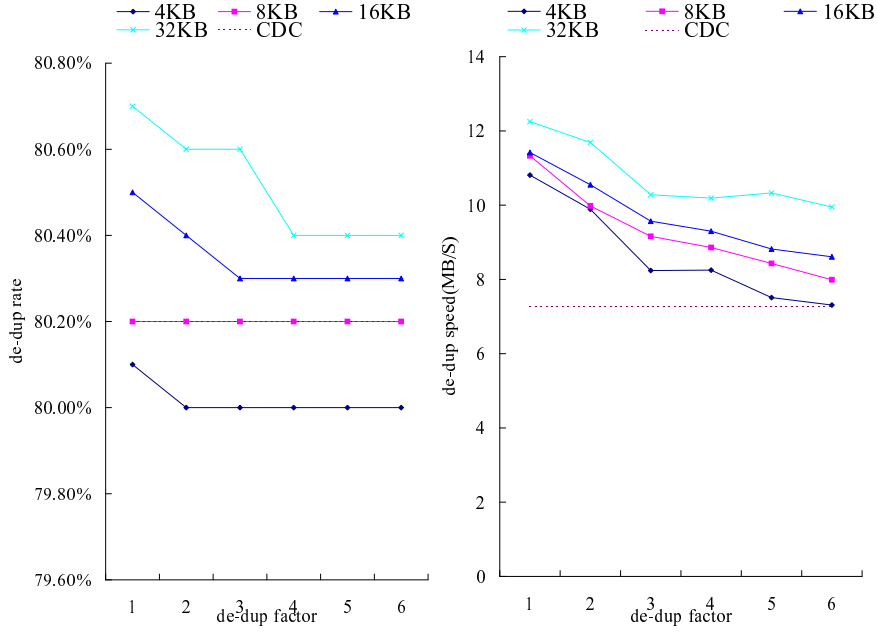


Figure 13. Testing on picture file type

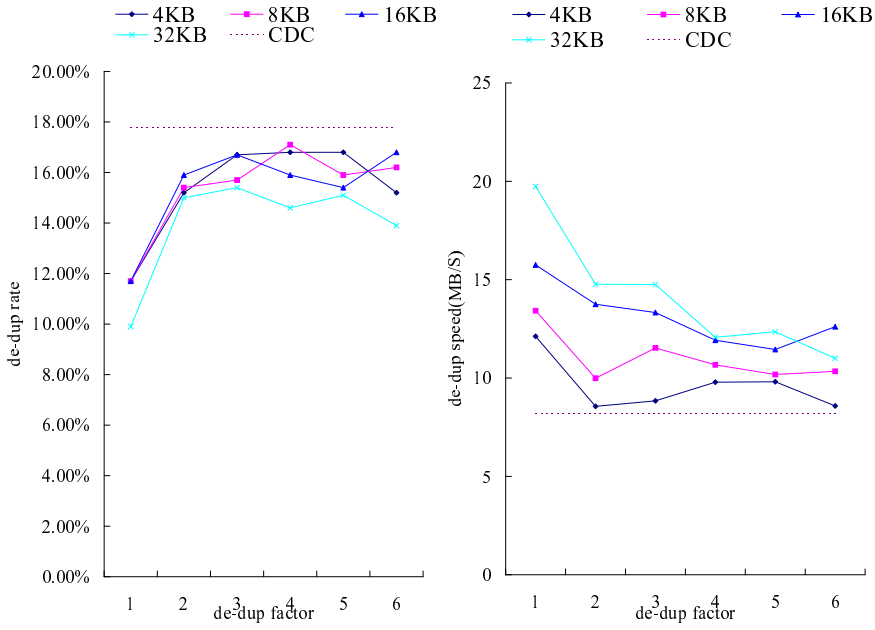


Figure 14. Testing on audio file type

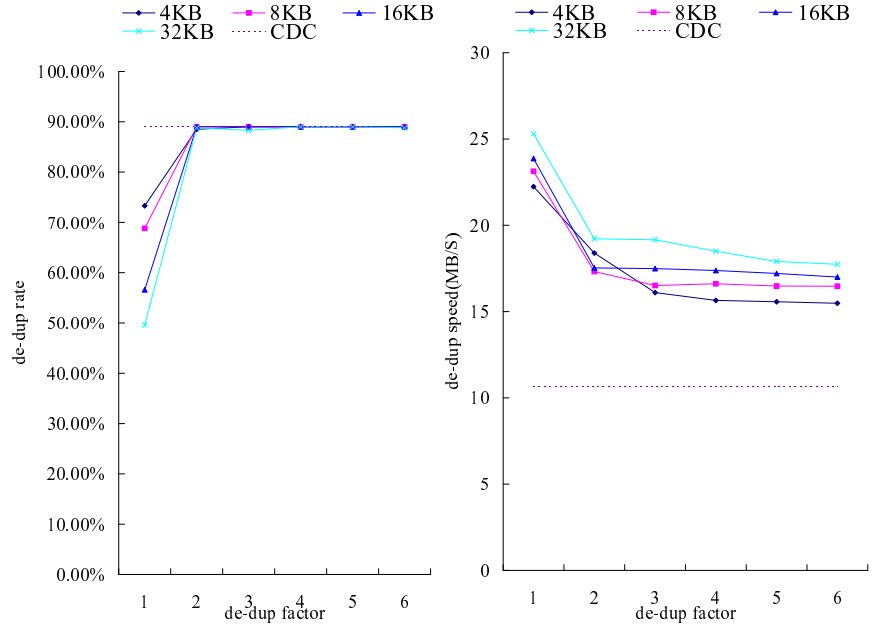


Figure 15. Testing on rar file type

The algorithm will turn into CDC when the de-dup factor is set to infinity, and it will become FSP if the acceleration factor is set big enough. Therefore, the FastCDC algorithm combines the advantages of FSP and CDC. It cannot only conduct de-dup detection rapidly for a new file, but also efficiently identify the modified parts of the modified files. Compared with SIS and FSP, both of which are suitable to application environment with no or little modification, CDC is suitable for the application environment with large number of modified data. FastCDC can be employed to both of the above environments effectively.

6.3.3 Performance Comparison of De-Duplication Algorithms

SIS, FSP, CDC and FastCDC are adopted to conduct full backup to the data in testing dataset II for fifteen consecutive times according to their corresponding month and the following testing results are acquired. In terms of the block-level oriented de-dup algorithm, its expected chunk length is 8 KB. The de-dup factor of FastCDC is set to 2 and the acceleration factor to 16 KB. The de-dup rate in Figure 16 a) actually refers to accumulated de-dup rate, which is obtained through dividing the total size of the present dataset to be backed up and the already backed up dataset by the size of the present dataset after having been de-duplicated.

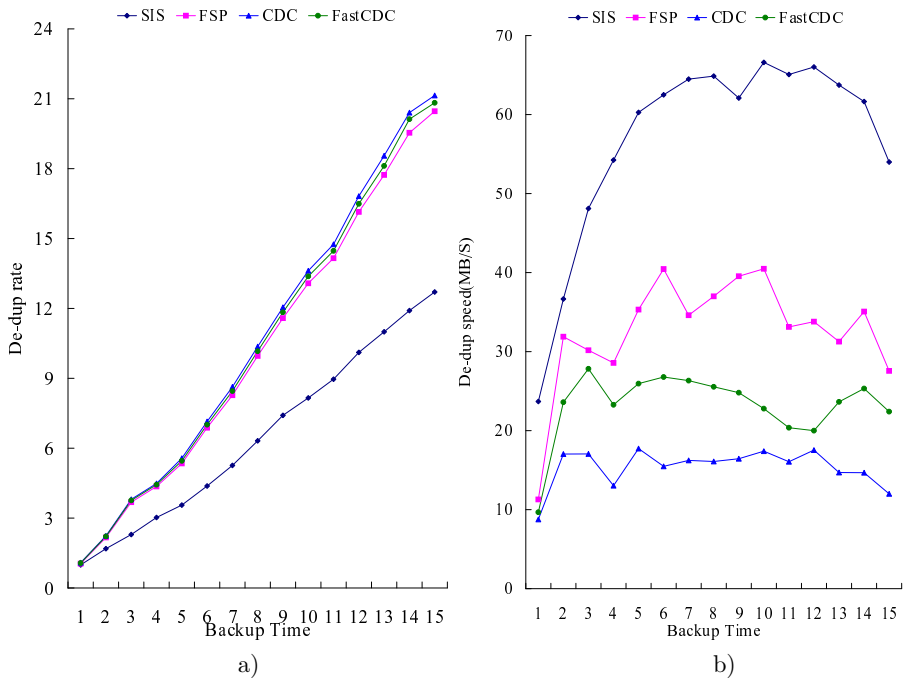


Figure 16. Performance testing on different de-duplication algorithms: a) De-dup rate, b) De-dup speed

Performance testing on different de-dup algorithms as the Figure 16 a) shows, with the increase of the backup times, the de-dup rates of all the algorithms will raise correspondingly, namely, the de-dup rate increases in proportion to the number of backup times. However, the de-dup rate of block-level algorithm is far higher than that of file-level algorithm. For instance, the de-dup rate of FSP CDC and FastCDC after fifteen consecutive backups can basically achieve 20 : 1, while the de-dup rate of SIS can only achieve 12 : 1. Moreover, after comparing CDC with FastCDC we can find that the de-dup rates of the two are basically the same. From Figure 16 b) we know that SIS boasts the highest de-dup rate, the highest of which is 66 MB/s and the average is 57 MB/s or so. The de-dup rate of FSP ranks first among all the block-level algorithms, with the highest being 40 MB/s and the average 32 MB/s or so; the de-dup rate of FastCDC is located in the mediate place between FSP and CDC, the highest approximating to 28 MB/s and the average 23 MB/s; while the de-dup rate of CDC is the lowest, with the highest rate being 17 MB/s and the average 15 MB/s. Compared with CDC, FastCDC can acquire the increase of de-dup speed by about 50 % at the expense of tiny de-dup rate loss.

7 CONCLUSIONS

This paper proposed a new de-dup technology-based system architecture to solve the problem of data redundancy during the process of data backup. To solve the problems of parameter setting and low de-dup speed of CDC, the self-adaptive de-dup algorithm and fast de-dup algorithm are proposed respectively. The experimental results show that the improved self-adaptive CDC can improve the de-dup rate by about 5 %, while FastCDC can obtain the increase of de-dup speed by 50 % to 200 % only at the expense of less than 3 % de-dup rate loss.

Acknowledgements

This work was supported by the projects of the National Key Technology R & D Program (Grant No. 2011BAH04B03, 2016YFB1000303) and the NSFC project (Grant No. 61572394). Part of the work was supported by the Marie Curie IRSES Actions of the European Union Seventh Framework Program (EU-FP7 Contract No. 318906). The authors wish to thank Xueqin Jiang, Xiaoxia Jiang, Yueguang Zhu, JiaJia Zhang and Huali Cui for useful discussions and help.

REFERENCES

[1] Beyer, M.: Gartner Says Solving "Big Data" Challenge Involves More Than Just Managing Volumes of Data. Gartner. Available on: `http://www.gartner.com/it/page.jsp?id=1731916`, Retrieved 13 July 2011.

[2] An Oracle White Paper in Enterprise Architecture-Information Architecture: An Architect's Guide to Big Data. August 2012.

[3] Ghemawat, S.—Gobioff, H.—Leung, S.-T.: The Google File System.

[4] Bolosky, W. J.—Corbin, S.—Goebel, D.—Douceur, J. R.: Single Instance Storage in Windows 2000. Proceedings of 4th USENIX Windows Systems Symposium (WSS '00), Vol. 4, 2000.

[5] Benson, M. L.—Shakib, D. A.: Single Instance Storage of Information. United States Patent 08/678, 995.

[6] Single Instance Storage in Microsoft Windows Storage Server 2003 R2. Microsoft Corp. Technical White Paper, published on May 2006.

[7] Bobbarjung, D. C.—Jagannathan, S.—Dubnicki, C.: Improving Duplicate Elimination in Storage Systems. ACM Transactions on Storage (TOS), Vol. 2, 2006, No. 4, pp. 424–448.

[8] Policroniades, P. I. C.: Alternatives for Detecting Redundancy in Storage Systems Data. Proceedings of the USENIX Annual Technical Conference (ATEC '04), 2004.

[9] Jain, N.—Dahlin, M.—Tewari, R.: TAPER: Tiered Approach for Eliminating Redundancy in Replica Synchronization. 4th USENIX Conference on File and Stora-Technologies (FAST 2005), 2005, pp. 281–294.

[10] Denehy, T. E.—Hsu, W. W.: Duplicate Management for Reference Data. IBM Research Report, RJ 10305 (A0310-017), 2003.

[11] Kruus, E.—Ungureanu, C.—Dubnicki, C.: Bimodal Content Defined Chunking for Backup Streams. Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST '10), 2010.

[12] Broder, A. Z.: Identifying and Filtering Near-Duplicate Documents. Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching (COM '00), London, 2000. Lecture Notes in Computer Science, Vol. 1848, 2000, pp. 1–10.

[13] Sun, G.-Z.—Dong, Y.—Chen, D.-W.—Wei, J.: Data Backup and Recovery Based on Data De-Duplication. 2010 International Conference on Artificial Intelligence and Computational Intelligence (AICI), 2010, pp. 379–382.

[14] Zhu, G.—Zhang, X.—Wang, L.—Zhu, Y.—Dong, X.: An Intelligent Data De-Duplication Based Backup System. 2012 15th International Conference on Network-Based Information Systems, 2012, pp. 771–776.

[15] Yang, T.—Feng, D.—Liu, J.—Wan, Y.: FBBM: A New Backup Method with Data De-Duplication Capability. International Conference on Multimedia and Ubiquitous Engineering (MUE 2008), 2008, pp. 30–35.

[16] Yang, T.—Feng, D.—Liu, J.—Wan, Y.—Niu, Z.—Ke, Y.: 3DNBS: A Data De-Duplication Disk-Based Network Backup System. IEEE International Conference on Networking, Architecture, and Storage (NAS 2009), 2009, pp. 287–294.

[17] Du, J.—Yu, H.—Zheng, W.: MassStore: A Low Bandwidth, High De-Duplication Efficiency Network Backup System. 2012 International Conference on Systems and Informatics (ICSAI), 2012, pp. 886–890.

[18] Ao, L.—Shu, J.-W.—Li, M.-Q.: Data De-Duplication Techniques. Journal of Software, Vol. 21, No. 5, May 2010, pp. 916–929.

[19] He, Q.—Li, Z.—Zhang, X.: Data De-Duplication Techniques. 2010 International Conference on Future Information Technology and Management Engineering (FITME), October 9–10, 2010, Vol. 1, pp. 430–433.

[20] Quinlan, S.—Dorward, S.: Venti: A new Approach to Archival Storage. Proceedings of the Conference on File and Storage Technologies (FAST '02), 2002, pp. 89–101.

[21] Sengar, S. S.—Mishra, M.: A Parallel Architecture for In-Line Data De-Duplication. 2012 Second International Conference on Advanced Computing and Communication Technologies, 2012, pp. 399–403.

[22] Hagwat, D.—Pollack, K.—Long, D. D. E.—Schwarz, T.—Miller, E. L.—Pâris, J. F.: Providing High Reliability in a Minimum Redundancy Archival Storage System. Proceedings of the 14th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2006), 2006, pp. 413–421.

[23] Lawrence, L.—Kristal, Y.—Pollack, T.—Darrell, D.—Long, E.: Deep Store: An Archival Storage System Architecture. Proceedings of the 21st International Conference on Data Engineering (ICDE 2005), 2005, pp. 804–815.

[24] Han, B.—Keleher, P.: Implementation and Performance Evaluation of Fuzzy File Block Matching. Proceedings of the 2007 USENIX Annual Technical Conference (USENIX 2007), 2007, pp. 199–204.

[25] Peng, C.—Wang, S.—Jia, Z.: Provide High Reliability for Duplicate Storage System with Erasure Code. Journal of Computer Research and Development, Vol. 48, 2011, pp. 1–6.

[26] Gu, Y.—Liu, C.—Sun, L.—Yan, B.—Wang, D.—Ju, D.: Reliability Provision Mechanism for Large-Scale De-Duplication Storage Systems. Journal of Tsinghua University (Science and Technology), 2010, Vol. 50, No. 5.

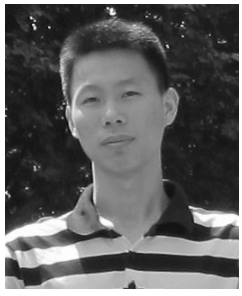
[27] Zhu, B.—Li, H.—Patterson, H.: Avoiding the Disk Bottleneck in the Data Domain De-Duplication File System. Proceedings of the 6th USENIX Conference on File And Storage Technologies (FAST '08), 2008.

[28] Lillibridge, M.—Eshghi, K.—Bhagwat, D.—Deolalikar, V.—Trezise, G.—Camble, P.: Sparse Indexing: Large Scale, Inline De-Duplication Using Sampling and Locality. Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST '09), 2009, pp. 111–123.

[29] Yang, T.—Jiang, H.—Feng, D.—Niu, Z.—Ke, Z.—Wan, Y.: DEBAR: A Scalable High-Performance De-Duplication Storage System for Backup and Archiving. 2010 IEEE International Symposium on Parallel and Distributed Processing (IPDPS), 2010, pp. 1–12.

[30] Eshghi, K.—Tang, H. K.: A Framework for Analyzing and Improving Content-Based Chunking Algorithms. Intelligent Enterprise Technologies Laboratory, HP Laboratories Palo Alto, HPL-2005-30(R.1), 2005.

Xingjun Zhang received his Ph.D. degree in computer architecture from Xi'an Jiaotong University, Xi'an, China, in 2003. From 1999 to 2005, he was Lecturer, Associate Professor in the Department of Computer Science and Technology of Xi'an Jiaotong University. From February 2006 to January 2009, he was Research Fellow in the Department of Electronic Engineering of Aston University, United Kingdom. He is currently Full Professor with the Department of Computer Science and Technology of Xi'an Jiaotong University. His interests include high performance computer architecture, computer network, storage system and high performance computing.



Guofeng Zhu received his B.Sc. degree in computer science and technology from Hainan University, Hainan, China, in 2010. He received his M.Sc. degree in computer science and technology from Xi'an Jiaotong University, Xi'an, China in 2013. He is currently a researcher in IBM China Systems and Technology Laboratory. His research interests include computer architecture and storage system.

Endong Wang received his B.Sc. and M.Sc. degrees from Tsinghua University in 1988 and 1991, respectively. He is now the President of Inspur Group. His research interests include high performance computing, computer architecture, parallel and distributed processing and microprocessor architecture.

Scott Fowler received B.Sc. from Minot State University, USA in 1998, M.Sc. from the University of North Dakota, USA in 2001 and Ph.D. from Wayne State University, USA in 2006, all degrees in computer science. During 2006–2010, he was Research Fellow in the Adaptive Communications Networks Research Group at Aston University, UK and Sony Ericsson R & D lab, UK, where the research focused on multiple services in Next Generation Networks (NGNs) in both wireless and wired, and the project team was composed of multi-disciplinary/multi-institutional partners from industry and academia. Since 2010, he has been Associate Professor at Linköping University, Sweden, and works with the Mobile Telecommunication (MT) group. He has served on several IEEE conferences/workshops as TPC to Chair. His research has been funded and supported by European Union Framework 7, Excellence Center at Linköping – Lund in Information Technology (ELLIIT), Ericsson and Ascom. His research interests include Quality of Service (QoS) support over heterogeneous networks, computer networks (wired, wireless), energy management, mobile computing, pervasive/ubiquitous, performance evaluation of networks and security. In 2012 he was awarded Visiting Professorship from the France Scientific Council to the University of Paris-Est Creteil (UPEC), France. He was a host for a Fulbright Specialist from the USA in 2011. He is a member of IEEE and ACM.

Xiaoshe Dong received his B.Eng. degree in computer hardware from Xi'an Jiaotong University, Xi'an, China, in 1985, and his M.Sc. and Ph.D. degrees in computer architecture, both from Keio University, Tokyo, Japan, in 1996 and 1999, respectively. He was Lecturer during the period 1987–1994 and Associate Professor during 1999–2003 in the Department of Computer Science and Engineering of Xi'an Jiaotong University, where he has been Full Professor with the Department of Computer Science and Engineering from 2003. His interests include high performance computer architecture and storage system.