ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)

https://doi.org/10.17559/TV-20200520034015
Original scientific paper

Data Deduplication Technology for Cloud Storage

Qinlu HE, Genqing BIAN*, Bilin SHAO, Weiqi ZHANG

Abstract: With the explosive growth of information data, the data storage system has stepped into the cloud storage era. Although the core of the cloud storage system is distributed file system in solving the problem of mass data storage, a large number of duplicate data exist in all storage system. File systems are designed to control how files are stored and retrieved. Fewer studies focus on the cloud file system deduplication technologies at the application level, especially for the Hadoop distributed file system. In this paper, we design a file deduplication framework on Hadoop distributed file system for cloud application developer. Proposed RFD-HDFS and FD-HDFS two data deduplication solutions process data deduplication online, which improves storage space utilisation and reduces the redundancy. In the end of the paper, we test the disk utilisation and the file upload performance on RFD-HDFS and FD-HDFS, and compare HDFS with the disk utilisation of two system frameworks. The results show that the two-system framework not only implements data deduplication function but also effectively reduces the disk utilisation of duplicate files. So, the proposed framework can indeed reduce the storage space by eliminating redundant HDFS file.

Keywords: cloud storage; data deduplication; distributed; file deletion; HDFS

1 INTRODUCTION

With the growth of social informatisation and the increase of the demand on information systems by enterprises and institutions, disaster recovery of information systems plays a key role in ensuring business continuity and data availability in the event of a disaster. Data shows that Baidu generates about dozens of petabytes of data every day, and Taobao generates more than 50TB of data every day. Therefore, enterprise data backup has a huge demand for data storage, which is making data centres face huge challenges. Data shows that Baidu generates about tens of petabytes of data every day, and Taobao generates more than 50 TB of data every day. Therefore, enterprise data backup has a huge demand for data storage, making data centres face a huge challenge. Although the cloud storage system with the distributed file system solves the problem of mass data storage, researches show that a large amount of duplicate data in the storage system has accelerated the growth of duplicate data. Generally, more than 90% of the data in the backup storage system is duplicate data [1], resulting in the shortage of storage space resources and network bandwidth in the system, and a rapid rise in data management costs. However, in order to curb excessive data growth, improve storage space utilisation, and reduce costs, deduplication technology has become a research topic that attracts much attention [2].

The advances in deduplication technology are based on the characteristics of the application. Maddodietal proposed data deduplication technology and analysed database applications, which try to reduce database storage consumption or process transactions online. For granularity, deduplication is applied at the sub-file or the entire file level [3-6], which improves the impact of space storage and performance trade-off evaluation issues. More fine-grained deduplication creates more space storage opportunities and causes a significant performance impact. When changing cloud systems, for example, file deduplication, the trade-off between space storage and performance impact is still a major issue in data deduplication technology. Chun-Ho Ng's focuses on the virtual memory image deletion technology on the open-source cloud, and under reasonable circumstances virtual memory image storage saves at least 40% of storage space [7]. SAM is a framework for a multi-layer semantic source deduplication cloud backup system. It has a deduplication rate as high as that based on global blocks, but its overhead is lower than that based on global blocks [8]. Shang & Li pointed out several shortcomings and corresponding solutions to the current cloud system data deletion technology work [9]. The challenging issue with cloud deduplication remains the trade-off between storage efficiency and performance.

This article addresses the huge pressure on storage devices caused by the current explosive growth of data, and focuses on the Hadoop-based distributed file system HDFS in cloud storage through the application of deduplication technology to improve disk space utilisation and upload time performance The research is carried out, and the main research works are shown as follows: (1) Based on HDFS, design, and implement efficient RFD-HDFS and FD-HDFS frameworks. The design goals of these two frameworks are to improve disk utilisation and file upload time. (2) In order to improve the utilisation of disk space, the data detection technology in the deduplication technology is used to compare the hash values of the files for redundancy elimination. (3) Based on the proposed RFD-HDFS and FD-HDFS frameworks, the disk utilisation and file upload time were tested separately and compared with the original HDFS to verify whether the disk utilisation and upload time have improved.

2 BACKGROUND AND MOTIVATION

In recent years, the deduplication technology is becoming more and more mature, but the data collision problems and the disk bottleneck problems are still big challenge tasks, which have attracted a lot of research attention.

2.1 Data Collision Problem

The essential feature of a data block is data fingerprint. In ideal state, each data block corresponds to a unique data fingerprint, which means different data blocks have different data fingerprints. The popular hash algorithms are

MD5, SHA1, SHA-256, SHA-512, Rabin Hash, etc. [10-12]. However, the problem is that different data blocks may produce the same data fingerprint. After comparison, the collision probability of MD5 [13] and SHA [14] series HASH functions is very low, so it is used as a common method for fingerprint calculation. MD5 and SHA1 are 128-bit, while SHA-1 has a lower collision rate than MD5, but the amount of calculation will increase significantly at the same time. In practical applications, we need to consider both performance and data security, and then select the appropriate method for calculation. Also, there is a more popular method for data fingerprint calculation with multiple hash algorithms. This method is highly secure, but system consumption is greatly increased. Due to the possibility of collision of data, dedupe technology is rarely used in critical data storage situations, because once a collision occurs, it will cause huge economic losses. Fu et al. [15] aims to minimise the time required by a deduplication process by parallelising the hash calculation process; however hashes are calculated for all blocks.

2.2 Disk Bottlenecks

Deduplication can be used to store backup data as well as non-backup data. The key issue with deduplication for non-backup data is still index disk bottlenecks. Also, non-backup data contains more non-duplicate data and requires more memory to store indexes, which makes this problem even more serious. Since 2011, some deduplication methods for storing non-backup data have been proposed [16], all of them, the most famous SSRC proposed HANDS. It predicts the working data set based on the access pattern of the data set and then dynamically prefetch data block signatures from disk. It can reduce a lot of memory usage. Therefore, it is one of the main problems based on data block deduplication [17]. In order to find the same data block quickly, all index blocks need to be put into memory. If there is fewer data in the system, you can directly put all index blocks into memory completely. Duggal et al. [18] made use of adjacency based resemblance detection for performing deduplication for backup systems. Their work was aimed to find the best possible blocks for delta compression in low overhead. Jin et al. [19] proposed D^3, a dynamic two-stage deduplication framework for distributed main storage, which is mainly improved by the following aspects to achieve a high deduplication rate. Xia et al. [5] improve the efficiency of the backup storage system by imparting both the inline and out-of-line deduplication. As deduplication causes fragmentation that degrades the read performance, Cao et al. [20] worked to improve the read performance.

2.3 HDFS Hash Function

The aforementioned technologies are widely used in enterprise storage device file servers, data blocks, NAS (RAID), and backup devices. Prior to HDFS, however, there was no special block state and network topology in Hadoop to ensure the reliability of backup instructions [21, 22]. That is, the condition of deleting duplicate files is only to detect similar or identical data. In addition to considering space and time consumption, we should also consider the reliability of the proposed algorithm or framework, and the system overhead required for the recovery process after frequent hardware errors. Therefore, we propose to use a streaming comparison function on HDFS for the entire file.

In the cloud environment without deduplication technology, Hadoop is widely used in big data storage and plays an important role [23-30]. The following uses a simple experiment as an example. Three identical files are stored on HDFS. Different files are named. On HDFS, each file is stored on at least three disks, which means three files with the same content are not deduplicated on HDFS. Consider a situation where millions of users share a popular Youtube video without deduplication, which could waste a very large amount of disk storage. To meet HDFS application equipment, deduplication schemes are proposed as necessary. Khan et al. [31] proposed a multi-level pattern-matching algorithm for deduplication of big data. MLPMA is implemented based on the similarity and locality of data, and the Bloom filter is applied to the deduplication cluster to achieve effective data deletion.

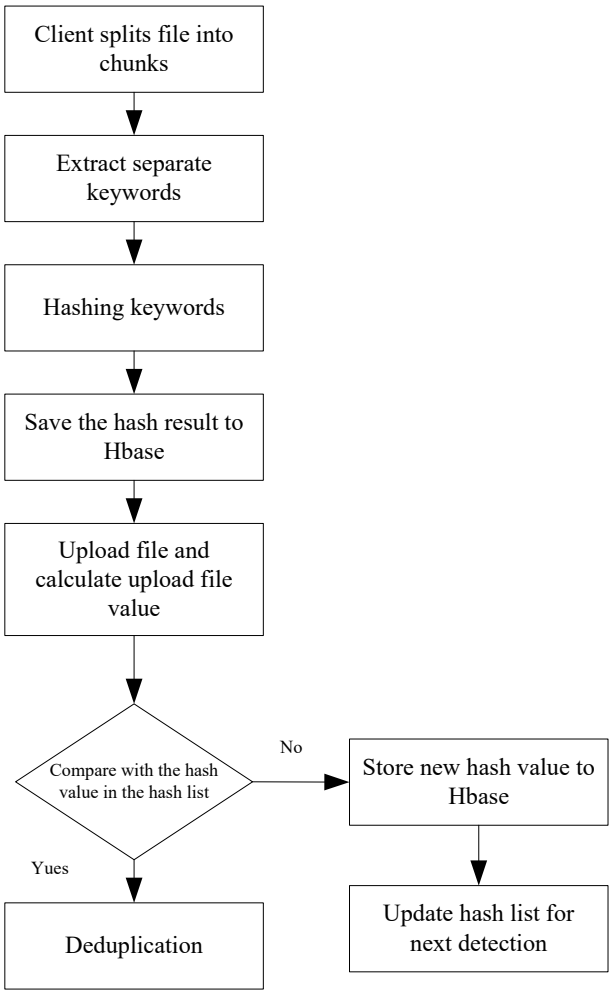

Figure 1 Deduplication process

3 DESIGN AND IMPLEMENTATION OF HDFS DATA DEDUPLICATION SYSTEM

3.1 Data Deduplication Design

Data deduplication occurs during the writing process, namely written on a document issued to a Client's request. First of all, comparing with similar data stored in the system to determine whether to duplicate data. Then, according to the result of decision save the file in actual

system, to improve the utilisation of the system space. The specific operation process of the data deduplication scheme used in this paper is shown in Fig. 1.

When a Client wants to write files to the system, for the files already stored in the storage system, their respective hash function values (usually MD5 or SHA-1) are calculated first; these hash function value groups are woven into a hash function value table, which is stored separately in Hbase. Finally, the hash value is queried in the local NameNode stored. If it exists, the data block is not stored, the uploaded file is deleted. If it does not exist, the data block is new, the system allocates space to store it, stores the hash value of the new file in Hbase, and updates the Hbase list. After the above steps, there may be that multiple files refer to the same data block, so when a client needs to delete the data corresponding to the file, but the other client does not have a file deletion request, the system cannot really delete the corresponding Data block, otherwise it will cause the data pointed by other files to be null data, affecting subsequent read requests of other clients.

3.2 Design of RFD-HDFS

In the design of RFD-HDFS, applications require accurate calculations without any errors, such as financial calculations, where errors are not allowed in the computing system. Because of the existence of a hash collision in SHA, the probability of a collision cannot be ignored (the probability of a collision is different depending on the algorithm). In binary comparisons, time and resources are spent comparing files to ensure the accuracy of the data. Therefore, use binary comparison or MapReduce cluster computing power to speed up file comparisons. At the same time, offline processing is adopted to reduce the waiting time of users. The first time the file is uploaded, it will be stored in a temporary storage pool. After that, the similar data detection program will start the file comparison task to check whether the file is duplicate. If the content is the same, the data deduplication task will be performed, or the request will be returned. If not, do the normal read and write task. In this article, stream comparisons are used to partially detect data fragments, and binary comparisons are made in the sequential serial method. The design framework of RFD-HDFS is shown in Fig. 2, and the following four steps are mainly used to determine whether the files are the same: (1) if the cache is hit in the Hash, the detection file size is defined to be the same. (2) If the cache is not hit in the Hash, the HDFS read and write command is executed. (3) cache hit, perform flow comparison, and detect whether the document is repeated by comparing file contents with the same hash value; (4) if the contents of the files are the same, the data will be deduplicated; if not, the files will be stored in the storage pool for the next hash collision detection and Hbase list update.

Fig. 2 shows the RFD-HDFS framework. A wrapper interface is defined as the HDFS client that provides a reliable deduplication function for the HDFS client and calls the HDFS to write a function in the background process. In addition to the surface wrapper, there are still three main parts in the framework, which are the hash generator, the storage pool, and the binary comparator. The HBase record table draws the hash value and the full path to the file. The hash generator is implemented using SHA2 and SHA3. Binary comparators can be hardware circuits or Hadoop MapReduce functions. A storage pool is the temporary file pool of background process. The binary comparator loads the file and makes the comparison in the background. All files are stored in the pool for logging information and rollback in case of failure.

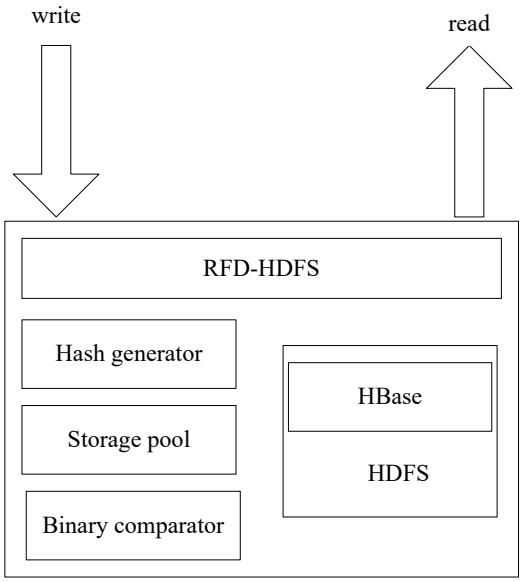

Figure 2 RFD-HDFS framework

3.3 Design of FD-HDFS

In applications, small errors are tolerated, such as web information extraction and lexical and semantic analysis. Repeated collisions are nearly impossible to occur, and the application's judgment results are not affected. A hash with the same fingerprint is considered a duplicate file. As a result, source duplication can reduce network bandwidth, save upload time, and even reduce the burden on HBase and primary nodes. In the case of web search software, Reiss and Reiss need to capture page files to HDFS every day. By comparing the HASH value to the HBase database, you can quickly know if the content of the site has changed. It can save a binary comparison time and retrieve the target directly from the SHA value generated at the source. If the source SHA does not change, all uploaded content is saved at the same time. If the hash generation algorithm is implanted from the source host, the load on the primary node and HMaster can be alleviated, and the original web page retrieval becomes incremental update retrieval, which cannot only save time and bandwidth but also reduce the load on the server. Applications can be fault-tolerant, like video sharing or many file shares, and FD-HDFS provides high performance and low storage consumption solution for file deletion. FD-HDFS design as shown in Fig. 3, calculated by hash generator upload file hash value for the first time, and stored in a comparison of the hash value of Hbase, see if there is if the hash value exists, requests for data deduplication or returns the command, if the hash value does not exist, then perform the normal read/write command, FD-HDFS is designed to save storage space when a hash collision occurs.

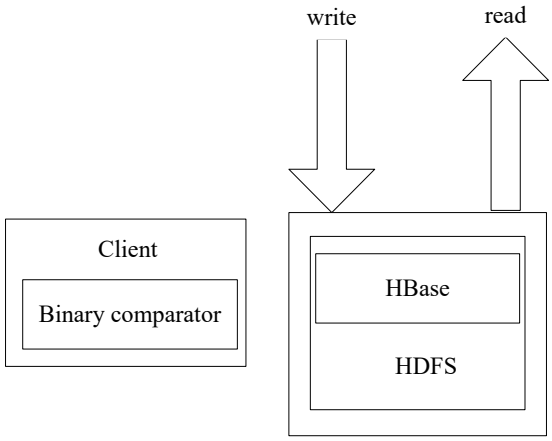

Figure 3 FD-HDFS frame

3.4 Realisation of Similar Data Detection

Data similarity detection is a process in the background and is responsible for receiving the same or similar data detection by the proposed deduplication framework. For both RFD-HDFS and FD-HDFS write instructions, if a similar record exists at HBase, it means that at least one file has the same contents stored in HDFS. If the existing file has the same path as the given file, there is no need to write the file to HDFS. That said, no more file transfers are required for FD-HDFS from the client to the HDFS server because HDFS can retrieve the file contents from itself. If the existing file path is different from the one given, RFD-HDFS will need to create a new record at HBase and store the file in a temporary file pool to prevent hash collisions and ensure the reliability of further file content retrieval. Also, for file read instructions, the FD-HDFS data similarity detector can help improve file download performance if the hash value is in the HBase table.

Simplify the RFD-HDFS implementation. When a file is written to RFD-HDFS, as shown in Fig. 2, write the file to HDFS and create a new record at HBase. Redundant data is deleted by performing data similarity detection in the background. This is what binary comparators are designed for, first detecting files with the same hash value from the storage pool and HDFS, and then comparing the contents of those files. If the contents of the files are the same, the binary comparator will delete the files from the storage pool for the purpose of redeleting the files. If the contents are different, the files will continue to be stored in the storage pool for further file retrieval requests and updated HBase records with the correct physical link to the file.

3.5 Data Routing

This paper presents a Data Routing Strategy using Semantics for distributed deduplication systems. File semantic information includes the information about a file, file ownership, document visitors, file size and other information. This information reflects the similarity degree of files. The higher the similarity degree, the higher the probability of duplicate data. In this paper, the data routing strategy of the distributed deduplication system is designed by using the file name as the semantic information of the file.

When the primary node receives the superblock generated by the backup server, the superblock data needs to be distributed to different heavy deletion points at the back end based on the file semantic information. The algorithm needs to establish and maintain a semantic data routing index table, which holds the semantic routing information related to the file name. The master node includes the semantic routing index and the container index. The semantic routing index includes the mapping between the file semantics and the target container. The data model is <filename, aim containterID, nodeID>. The reason why the algorithm chooses to migrate the smallest container instead of the largest container is that migrating the largest container is more likely to overload other nodes and cause a load balancing process for a series of nodes. After the data migration process is complete, the storage node needs to return the metadata of the migrated files to the master node. DR algorithm is shown as below:

DR Algorithm:
Input: filename, chunkingResults, NumOfContainers
Output: ANode
1: AContainer = SRIndex. Find(filename);
2: if AContainer == NULL then
3: MChunkID = FMChunkID(chunkingResults)
4: AContainer = MChunkID % NumOfContainers;
5: SRIndex.insert(filename, AContainer);
6:end if
7: ANode = CIndex.find(aContainer);
8: while sizeOfNode(ANode) > (ANodeSize()(1+MThreshold)) do*
//The ***MThreshold*** *is used to control the frequency of load balancing operations. The threshold is usually set to 0.05. That is, when the storage load of a node is 1.05 times the average storage load in the cluster, the data of the node needs to be migrated to other nodes*
9: MNode = FSNode();
10: SContainer = FSContainer(ANode);
11: MContainer(SContainer, MNode);
12: MChunkIndex(SContainer, MNode);
13: end while
14: return ANode;

DR algorithm firstly determines the ID of the target container through the routing index, and then queries the target node through the container index. The storage node searches for the duplicate data block by searching the block index and deduplicates the data. A single container stores multiple files that are only routed to the target container. Containers are used to load balance the deduplication cluster.

4 EVALUATION

4.1 Test Environment

Each Hadoop node setting is listed in Tab. 1. Tab. 2 shows the samples and architecture of HBase. Tab. 3 shows the number of files, total file size, and percentage of duplicate files. The experiment was run on a virtual machine, the operating system used CentOS 7.5, the hardware disk size was 1 TB, the rotation speed was 5400 rpm, and the data set was a different version of the same

open-source search platform package under Apache. All nodes are deployed on the same host.

The data set of the experiment is a Linux source code document that contains a series of versions from 1.3.0 to 2.5.75, with a total of 460 versions of data. The feature of the multi-version Linux source code documentation is that the data set has high redundancy, and most of the files in the data set are also small files. Through deduplication experiments on Linux source code data sets, experimental data is obtained.

Table 1 Hadoop node settings

HostName	OS	IP	HBase role	Process
NameNode	CentOS 7.5	10.13.32.190	HMaster	FD.jar&RFD.jar
DateNode1	CentOS 7.5	10.13.32.13	HRegion	
DateNode2	CentOS 7.5	10.13.32.191	HRegion	
DateNode3	CentOS 7.5	10.13.32.24	HRegion	

Table 2 HBase table samples and patterns

Row Key	Timestamp	File attributes			
		Route	Authority	Size	Time
File HASH	T1	Test/apache-slor-4.0.0.tagz	-rwxr-xr-x	200M	T11
File HASH	T2	Test/apache-slor-4.0.1.tagz	-rwxr-xr-x	200M	T12
File HASH	T3	Test/apache-slor-4.0.2.tagz	-rwxr-xr-x	200M	T13

4.2 Test Results and Analysis

4.2.1 DR Algorithm Analysis

Comparing with the single-node deduplication system, DR algorithm, DS algorithm MCS algorithm three distributed deduplication systems, when the DR algorithm data migration threshold is set to 0.05, in different sizes of deduplication cluster, several algorithms duplicate. The comparison between data deletion rates is shown in Fig. 4. In Fig.4, the deduplication rate of the DR and DS algorithms in the deduplication system with different numbers of nodes is the closest. The reason is that the smallest unit of data migration is a container rather than a single file. The DR algorithm can determine similar files. It is routed and stored in the same container. This method can ensure that after data migration is completed, similar data is still stored in the same node. It can be seen from the data that the DRSSLB algorithm has no effect on the deduplication rate.

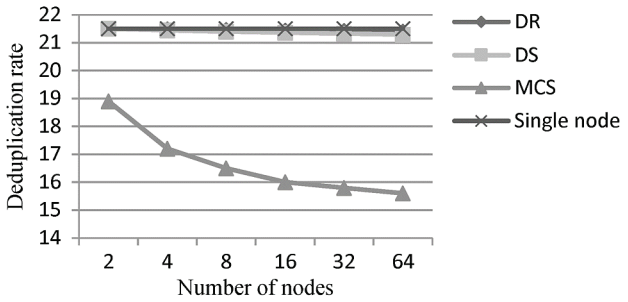

Figure 4 Comparison of algorithm deduplication rates

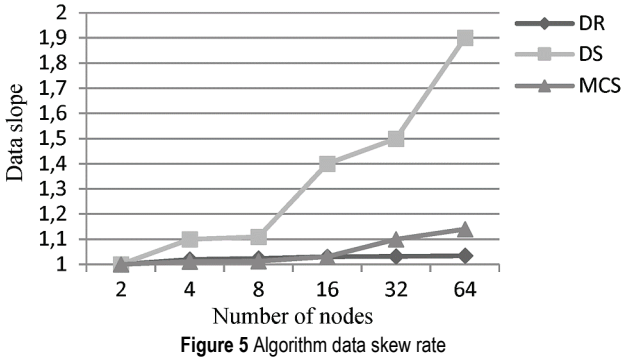

Figure 5 Algorithm data skew rate

Fig. 5 shows the comparison of DR, DS and MCS data skew rates. The data migration threshold of the DR algorithm is 0.05. The data in Fig. 4 shows that for different numbers of nodes, the slope of DR is lower than DS. When the number of nodes is greater than 16, the data slope rate of DR is lower than that of MCS. It can be seen from the experimental data that the DR algorithm has a good load balancing effect.

4.2.2 Disk Consumption Evaluation

In our experiment, 2000 files were uploaded to HDFS. The test scenario is repeatedly uploading files of different sizes to HDFS and recording the amount of disk space used. The results are shown in Tab. 3 and illustrated in Fig. 6.

Table 3 Test data

HostName	File size / MB	Number of files	Deduplication rate / %	File size * 3
USER01	7.2	18	66	21.6
USER02	14.1	25	76	42.3
USER03	6.3	30	30	18.9
USER04	8.3	22	59	24.9
USER05	68.9	24	58	206.7
USER06	10	22	13	30
USER07	23.2	38	42	69.6
USER08	101.6	144	41	304.8
USER09	70.3	120	50	210.9
USER10	15.6	33	57	46.8
USER11	41.9	62	38	125.7
USER12	94.3	164	17	282.9
USER13	61.1	108	26	183.3
USER14	25.3	38	73	75.9
USER15	34.7	62	74	104.1
USER16	24.4	42	83	73.2
USER17	3.6	15	33	10.8
USER18	20.1	31	61	60.3
USER19	12.1	20	75	36.3
USER20	3.7	18	72	11.1
Total	646.7	1036	40.9	1940.1

The abscissa is the record of space use time, and the ordinate is the size of space. The unit is GB. After 20 users uploaded their files, RFD-HDFS and FD-HDFS have less storage space than normal HDFS. As shown in Tab. 3, the total number of duplicate test files accounted for 40.9% of the total number of uploaded test files, and the disk utilisation rate reached only 59.1%. The results show that

only 46% of HDFS disk space is used for RFD-HDFS and FD-HDFS. The experimental results show that the system can effectively reduce the disk utilisation of duplicate files.

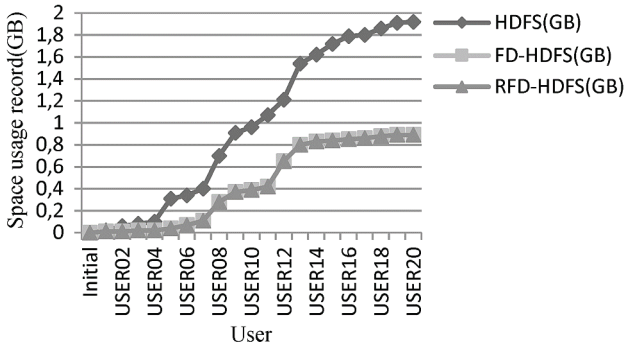

Figure 6 Comparison of HDFS and the proposed deduplication framework

Table 4 Test results

Time	HDFS type		
	HDFS / GB	FD-HDFS / GB	RFD-HDFS / GB
Initial	0.00	0.00	0.00
USER01	0.02	0.01	0.01
USER02	0.06	0.01	0.01
USER03	0.08	0.02	0.02
USER04	0.10	0.02	0.02
USER05	0.31	0.04	0.04
USER06	0.34	0.07	0.07
USER07	0.40	0.11	0.11
USER08	0.70	0.28	0.28
USER09	0.91	0.37	0.37
USER10	0.96	0.39	0.39
USER11	1.07	0.42	0.42
USER12	1.21	0.65	0.65
USER13	1.54	0.80	0.80
USER14	1.62	0.83	0.83
USER15	1.72	0.84	0.84
USER16	1.79	0.85	0.85
USER17	1.80	0.86	0.86
USER18	1.86	0.88	0.88
USER19	1.91	0.89	0.89
USER20	1.92	0.89	0.89

4.2.3 File Upload Performance Evaluation

In the experiment, the performance of RFD-HDFS and FD-HDFS is very similar because RFD-HDFS uses background processing file deduplication, so only FD-HDFS performance evaluation. When only data-like detectors work, performance may be affected. This experiment uses six files for testing. As shown in Tab. 5, the files vary in size, ranging from 131 MB to 4989 MB. The test scenario is uploading files one by one, and recording the time required for each file upload.

Table 5 Performance evaluation of test data

File ID	File size / MB
File 0	131
File 1	1001
File 2	2051
File 3	3024
File 4	3992
File 5	4989

The performance evaluation results are shown in Tab. 7. According to the data in the table, the upload performance of FD-HDFS and HDFS is very similar. Fig. 5 shows the upload time trend. The abscissa is the file size, the unit is MB, and the ordinate is the upload time, the unit is s. It shows that the upload time of HDFS is increasing linearly. The upload time of HDFS is slightly longer than that of FD-HDFS, but it is very close. Only when the uploaded file is relatively large, there is a certain gap. Therefore, it can be concluded that the deduplication process has little effect on upload performance, and the total consumption is negligible.

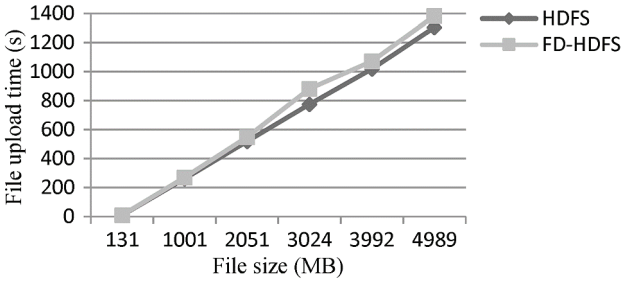

Figure 7 Performance comparison between HDFS and FD-HDFS

Table 6 Performance evaluation test results

File size	DFS file upload time / s	
	HDFS	FD-HDFS
131	9	10.4
1001	260	270.6
2051	520	548.3
3024	774	881
3992	1020	1073
4989	1305	1383

This chapter first introduces the test environment, including machine configuration, HBase samples and architecture. Then, the disk consumption and file upload ability of the designed deduplication framework were tested and analysed. The final results show that the performance of the proposed RFD-HDFS and FD-HDFS frameworks is very similar. Compared with the original HDFS, it effectively reduces the disk utilisation of duplicate files, but file upload is not affected by deduplication. HDFS's file upload performance is very similar.

5 CONCLUSIONS

This paper focuses on Hadoop-based distributed file system HDFS in improvement of the utilisation of disk space and the performance of upload time through the application of deduplication technology. Based on the proposed RFD-HDFS and FD-HDFS framework, the disk utilisation and file upload time were tested separately, and a comparison test was performed with the original HDFS to verify whether the disk utilisation and file upload time have been improved. The display effectively improves disk utilisation, but file uploads are not affected by deduplication. Therefore, the new architecture has similar performance to the original HDFS.

As the research on deduplication based on cloud storage is rich in content and is in the development stage, in view of the limitation of time and my knowledge level, some problems have not been solved well. These issues include:

(1) Client load problem. The deduplication function in this system is mainly concentrated on the client, so the client load is large, so the client load problem needs to be resolved in the next research work.

(2) System performance problems. The system test shows that although this system implements the deduplication function, the deduplication function is time consuming and system resources consuming, which causes the system's read and write performance to decline. The next step is to study this issue.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (61872284); Industrial field of general projects of science and Technology Department of Shaanxi Province (2020GY-012); Special Scientific Research Project of Shaanxi Provincial Department of Education (18JK0466); "Thirteenth Five-Year" National Key R & D Program Project (Project Number: 2019YFD1100901); Talent Fund Project of Xi'an University of Architecture and Technology (RC1707); Youth Fund Project of Xi'an University of Architecture and Technology (QN1726); Natural Science Project of Xi'an University of Architecture and Technology(ZR18050).

6 REFERENCES

[1] Duggal, A., Jenkins, F., Shilane, P., Chinthekindi, R., Shah, R., & Kamat, M. (2019). Data domain cloud tier: Backup here, backup there, deduplicated everywhere! *2019 USENIX Annual Technical Conference (ATC), 19*, 647-660.

[2] Harnik, D., Hershcovitch, M., Shatsky, Y., Epstein, A., & Kat, R. (2019). Sketching volume capacities in deduplicated storage. *ACM Transactions on Storage (TOS), 15*(4), 1-23. https://doi.org/10.1145/3369737

[3] Harnik, D., Khaitzin, E., & Sotnikov, D. (2016). Estimating unseen deduplication—From theory to practice. *14th {USENIX} Conference on File and Storage Technologies (FAST), 16*, 277-290.

[4] Matsuzawa, K., Hayasaka, M., & Shinagawa, T. (2018). The quick migration of file servers. *Proceedings of the 11th ACM International Systems and Storage Conference*, 65-75. https://doi.org/10.1145/3211890.3211894

[5] Xia, N., Tian, C., Luo, Y., Liu, H., & Wang, X. (2018). UKSM: Swift memory deduplication via hierarchical and adaptive memory region distilling. *16th USENIX Conference on File and Storage Technologies (FAST), 18*, 325-340.

[6] Cao, Z., Wen, H., Wu, F., & Du, D. H. (2018). ALACC: Accelerating restore performance of data deduplication systems using adaptive look-ahead window assisted chunk caching. *16th USENIX Conference on File and Storage Technologies (FAST), 18*, 309-324.

[7] Allu, Y., Douglis, F., Kamat, M., Prabhakar, R., Shilane, P., & Ugale, R. (2018). Can't we all get along? Redesigning protection storage for modern workloads. *2018 USENIX Annual Technical Conference (ATC), 18*, 705-718.

[8] Douglis, F., Duggal, A., Shilane, P., Wong, T., Yan, S., & Botelho, F. (2017). The logic of physical garbage collection in deduplicating storage. *15th USENIX Conference on File and Storage Technologies (FAST), 17*, 29-44.

[9] Agarwal, B., Akella, A., Anand, A., Balachandran, A., Chitnis, P., Muthukrishnan, C., Ramjee, R., & Varghese, G. (2010). Endre: An end-system redundancy elimination service for enterprises. *7th USENIX Conference on Networked Systems Design and Implementation (NSDI), 10*, 419-432.

[10] Sun, Z., Kuenning, G., Mandal, S., Shilane, P., Tarasov, V., & Xiao, N. (2016). A long-term user-centric analysis of deduplication patterns. *2016 32nd Symposium on Mass Storage Systems and Technologies (MSST)*, 1-7. https://doi.org/10.1109/MSST.2016.7897080

[11] Anderson, E., Hall, J., Hartline, J., Hobbs, M., Karlin, A. R., Saia, J., Swaminathan, R., & Wilkes, J. (2001). An experimental study of data migration algorithms. *5th International Workshop on Algorithm Engineering (WAE), 1*, 145-158. https://doi.org/10.1007/3-540-44688-5_12

[12] Anderson, E., Hobbs, M., Keeton, K., Spence, S., Uysal, M., & Veitch, A. C. (2002). Hippodrome: Running circles around storage administration. *1stUSENIX Conference on File and Storage Technologies(FAST), 2*, 175-188.

[13] Anderson, E., Kallahalla, M., Spence, S., Swaminathan, R., & Wang, Q. (2002). Ergastulum: Quickly finding near-optimal storage system designs. *HP Laboratories*.

[14] Bessani, A., Correia, M., Quaresma, B., André, F., & Sousa, P. (2013). DepSky: Dependable and secure storage in a cloud-of-clouds. *ACM Transactions on Storage (TOS), 9*(4), 1-33. https://doi.org/10.1145/2535929

[15] Fu, M., Han, S., Lee, P. P., Feng, D., Chen, Z., & Xiao, Y. (2018). A Simulation Analysis of Redundancy and Reliability in Primary Storage Deduplication. *IEEE Transactions on Computers, 67*(9), 1259-1272. https://doi.org/10.1109/TC.2018.2808496

[16] Bhagwat, D., Eshghi, K., Long, D. D., & Lillibridge, M. (2009). Extreme binning: Scalable, parallel deduplication for chunk-based file backup. *2009 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems*, 1-9.

[17] Chen, F., Luo, T., & Zhang, X. (2011). CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives. *9thUSENIX Conference on File and Storages Technologies(FAST), 11*, 77-90.

[18] Duggal, A., Jenkins, F., Shilane, P., Chinthekindi, R., Shah, R., & Kamat, M. (2019). Data Domain Cloud Tier: Backup here, backup there, deduplicated everywhere!. *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 647-660. https://doi.org/10.1109/MASCOT.2009.5366623

[19] Yin, J., Tang, Y., Deng, S., Li, Y., & Zomaya, A. Y. (2017). D^3: A Dynamic Dual-Phase Deduplication Framework for Distributed Primary Storage. *IEEE Transactions on Computers, 67*(2), 193-207. https://doi.org/10.1109/TC.2017.2743199

[20] Cao, Z., Wen, H., Wu, F., & Du, D. H. (2018). {ALACC}: Accelerating restore performance of data deduplication systems using adaptive look-ahead window assisted chunk caching. *16th {USENIX} Conference on File and Storage Technologies ({FAST} 18)*, 309-324.

[21] Chen, L. C., Wei, Z. P., Cui, Z. H., Chen, M.Y., Pan, H. Y., & Bao, Y. G. (2014). CMD: Classification based memory deduplication through page access characteristics. *10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE), 49*(7), 14. https://doi.org/10.1145/2576195.2576204

[22] Clements, A. T., Ahmad, I., Vilayannur, M., & Li, J. (2009). Decentralized Deduplication in SAN Cluster File Systems. *USENIX Annual Technical Conference, 9*, 101-114.

[23] Dantzig, G. B. (1963). *Linear programming and extensions*. Princeton university press. https://doi.org/10.7249/R366

[24] Rad S. M. & Nejad, M. B. (2019). New analog processing technique in multichannel neural signal recording with reduce data rate and reduce power consumption. *Traitement du Signal, 36*(2), 133-137. https://doi.org/10.18280/ts.360202

[25] Debnath, B. K., Sengupta, S., & Li, J. (2010). ChunkStash: Speeding up inline storage deduplication using flash memory. *2010 USENIX Conference on USENIX Annual Technical Conference (ATC), 10*, 1-16.

[26] Dong, W., Douglis, F., Li, K., Patterson, R. H., Reddy, S., & Shilane, P. (2011). Tradeoffs in scalable data routing for deduplication clusters. *9th USENIX Conference on File and Stroage Technologies (FAST), 11*, 15-29.

[27] Douglis, F., Bhardwaj, D., Qian, H., & Shilane, P. (2011). Content-aware load balancing for distributed backup. *25th International Conference on Large Installation System Administration (LISA), 11*, 1-18.
[28] Douglis, F., Duggal, A., Shilane, P., Wong, T., Yan, S., & Botelho, F. (2017). The logic of physical garbage collection in deduplicating storage. *15th USENIX Conference on File and Storage Technologies (FAST), 17*, 29-44.
[29] Wang, C., Wang, J. H., Sun, X. H., & Wang, F. S. (2018). A novel soil nutrient classification method based on hadoop platform. *Revue d'Intelligence Artificielle, 32*(S1), 25-40. https://doi.org/10.3166/ria.32.s1.25-40
[30] Li, Y. (2018). Design and implementation of intelligent travel recommendation system based on internet of things. *Ingénierie des Systèmes d'Information, 23*(5), 159-173. https://doi.org/10.3166/isi.23.5.159-173
[31] Khan, A., Lee, C. G., Hamandawana, P., Park, S., & Kim, Y. (2018). A robust fault-tolerant and scalable cluster-wide deduplication for shared-nothing storage systems. *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 87-93. https://doi.org/10.1109/MASCOTS.2018.00016

Contact information:

Qinlu HE
School of Information and Control Engineering,
Xi'an University of Architecture and Technology,
Xi'an 710043, China
E-mail: luluhe8848@hotmail.com

Genqing BIAN
(Corresponding author)
School of Information and Control Engineering,
Xi'an University of Architecture and Technology,
Xi'an 710043, China
E-mail: bgq_000@163.com

Bilin SHAO
School of Management,
Xi'an University of Architecture and
Technology, Xi'an 710043, China
E-mail: shaobilin@sina.com

Weiqi ZHANG
School of Information and Control Engineering,
Xi'an University of Architecture and Technology,
Xi'an 710043, China
E-mail: zhangwq_69@163.com