PROCEEDINGS OF SCIENCE

A Method to Improve the Performance for Storing Massive Small Files in Hadoop

Tong Zheng
East China University of Science and Technology
Shanghai, 200237, China
*E-mail:*skdzhengtong@163.com

Weibin Guo[1, 2]
East China University of Science and Technology
Shanghai, 200237, China
*E-mail:*gweibin@ecust.edu.cn

Guisheng Fan
East China University of Science and Technology
Shanghai, 200237, China
*E-mail:*gsfan@ecust.edu.cn

As a new open source project, Hadoop provides a new way to store massive data. Because of high scalability, low cost, good flexibility, high speed and strong fault tolerance performance, it has been widely adopted by the internet companies. However, the performance of Hadoop will reduce significantly once it is used to handle massive small files. As a result, this paper proposes a new scheme to merge small files, which occupymuch memory in NameNode, into large files and establish the mapping relationship between small files and large files, and then store the mapping information in HBase. In order to improve the reading performance, the scheme provides a prefetching mechanism by analyzing the access logs and putting the metadata frequently accessed merge files in the client's memory. The experiment results show that this scheme can efficiently optimize small files storage in HDFS, thus reduce the overload of NameNode and improve the performance of file access.

CENet2017
22-23 July 2017
Shanghai, China

1Corresponding Author

2This study is supported by the National Natural Science Foundation of China(Grant No.: 61672227)

PoS(CENet2017)022

1.Introduction

With the development of the network information technology, large amount of data are generated every day. Hadoop is a popular distributed framework which mainly consists of a high-performance distributed computing platform MapReduce [1] and a distributed storage framework HDFS (Hadoop Distributed File System) [2]. Because of its high scalability, low cost, high flexibility, high speed and strong fault tolerance, HDFS is widely used in the field of storing massive data. The study finds that microblog, QQ, Facebook and other large social networking softwares generate daily social data at the level of PB, including logs, photographs, short messages, etc., whose size is mainly at the level of KB. HDFS is designed to store the oversized files originally, and it is suitable to store and access data with high throughput, while it is not suitable to store the massive small files.

HDFS is composed of a NameNode and multiple DataNode, and the metadata of each file stored in HDFS exists in the memory of the NameNode. When large amount of small files are stored in HDFS, much more metadata will be stored in NameNode and take up much memory, which therefore decreases the performance of the system. And the number of stored files will be limited by the size of the NameNode memory. In addition, when reading a large number of small files, many requests need to be sent to the NameNode for the purpose of obtaining the file metadata, and then converted between multiple DataNode to find and read small files. The time occupied by reading or writing small files is far less than the time spent on the preparatory work, such as RPC communication, establishing a socket connection and addressing. It greatly decreases the performance of HDFS. So the study on how to solve the problem of storing small files in HDFS has become a popular research direction.

In this paper, we propose a new scheme to solve the problem of small files by merging small files into large files and establish the mapping relationship between small files and large files. In addition, we also design a prefetching mechanism by analyzing the access logs and putting the metadata frequently accessed merge files in the client's memory, thereby improving the reading performance.

The rest of this paper is organized as follows. Section 2 introduces the related work. Section 3 describes the architecture of the small files to processand analyze the algorithm about the process. The numerical results and HDFS thoroughly compared with original ones are given in Section 4. Finally, some conclusions and the shortcomings are drawn in Section 5.

2.Related Work

In order to solve the problems of small files, Hadoop provides three solutions: Hadoop Archive [3], Sequence File [4] and CombineFileInputFormat [5]. However, these methods also bring other new problems. For example, after merging small files by Hadoop Archive technology, the source files will not be deleted automatically. What's more, HAR file does not support compression and cannot be modified once created. Sequence File does not create a mapping between a small file and the large file and it needs to traverse the entire sequence file when looking for a small file.

Some scholars have studied this problem from different perspectives. Zhang et al. proposed a method that store small files in accordance with the size range of files [6]. Files that are less than 50 KB are stored in HBase, and files that range from 50 K to 1 MB are stored in the merge method which was proposed by the author, and files larger than 1 MB are stored by

using AVRO . Patel and Mehta presented a method by which the associated small files are merged into large files, and they also established a prefetch and index mechanism. But, it is necessary to provide small files name and large files name when reading small files [7]. In order to improve the metadata management of HDFS, Vorapongkitipun et al. proposed a New Hadoop Archive (NHAR) by a re-designed indexing mechanism without changing HDFS architecture in accessing file performace. [8]. Considering the correlation between small files, Dong et al. merged all the files belonging to the same center of PPT courseware to reduce the metadata burden on the NameNode and use the prefetching technology to improve the accessing efficiency [9]. Liu and his collaborator combined the characteristics of WebGIS to merge the adjacent files based on the physical location and build an index for each file [10]. Taobao and Facebook, in order to efficiently store the small files, have developed the Taobao File System (TFS) and Haystack storage system [11].

Although the above schemes can solve the problem of storing small files in certain fields, such as PPT courseware and image, etc., they cannot be applied to other fields, such as audio and video, etc.. And some other methods must be used to provide the small files name and the merged files name to access small files, which will greatly reduce the performance of Hadoop.

3.Small Files Processing Scheme

The main idea of this paper is to merge the small file, that is, we will merge some small files into a large file and then upload the large file to HDFS instead of uploading each small file individually. During the merging process, we establish the mapping relationship between small files and large files. The mapping index is saved to HBase when the large file is uploaded. In order to improve the efficiency of reading small files, and the scheme also supports prefetching, which puts some metadata of frequently accessed small files into client's memory according to the access logs. The overview of the scheme is shown in Figure 1.

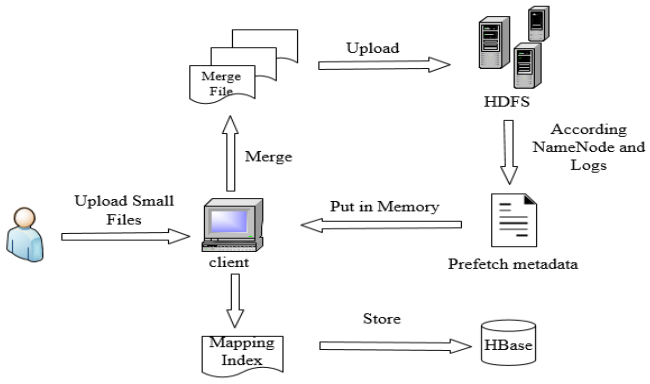

Figure 1: Overview of the Scheme

3.1The Proposed Scheme

This scheme is mainly used to process small files and does not take large files into accountI. It mainly consists of three processes: the merging small files, the establishing mapping index and the prefetching. Detailed description of the three processes is shown as follows.

3.1.1Merging Small Files

When it needs to upload the small files to HDFS, the client creates a temporary file in the memory firstly. Then, small files are merged into the temporary file. If the condition is fulfilled,

the size of the temporary file reaches the block size (default 64 MB) or the temporary file has been reserved in the memory for the time of T1 and thus the system will upload the temporary file to HDFS. The time stamp is used to name the merge file and upload it to HDFS, e.g., 20170417132345.

T1 is designed to prevent the next time period when no small files need to be uploaded or the network fails, or system and other reasons leading to the client failure in receiving other small files, which will result in the temporary files saved in memory and cannot be uploaded to HDFS timely. T1 can be set according to the actual situation. In practice, the size of the merge file cannot be exactly equal to that of the block, and the temporary file should be uploaded when it meets the conditions that the sum of its size and the size of next small file are larger than the block size.

3.1.2Establishing the Mapping Index

At the beginning of merging small files, an index file is created which is used to store the small files name, size and the offset in the merge file. The mapping between the index and the merge file is shown in Figure 2.

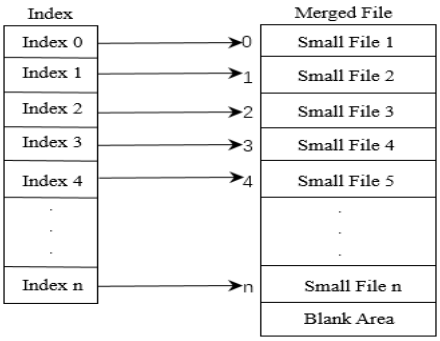

Figure 2: Mapping Between the Index and the Merge File

When the merge file is uploaded to HDFS, the content of the index file and the merge file name are saved to HBase. HBase is a non-relational and a distributed database for column storage. It has the advantage of high-performance concurrent read and write operations and the efficient access rate for small files [12]. HBase mainly contains the row key, column family and time stamp. The row key is the primary key and it is used to retrieve the records, and each column in HBase belongs to a column family.

In this scheme, the name of the small file is saved as Row Key. The table also contains a column family called content, which includes the merged file name, the small file size and the offset of small files in the merge file. The mapping index stored in HBase is shown in Table 1. The merge files name and the location of small files in merge files can be acquired according to it.

Row Key	Time Stamp	Column Family (content)		
		Merge File Name	Small File Size	Offset
Small file 1	T1	Merge file 1	100 K	0
Small file 2	T2	Merge file 2	97 K	100
...	...	...	...	...
Small file i	Ti	Merge file j	...	...
...	...	...	...	...
Small file n	Tn	Merge file k	...	...

Table 1: The Mapping Table Structure

3.1.3Prefetching

In order to improve the efficiency of reading small files, the scheme analyzes the access logs of HDFS periodically and takes out the last N access records to calculate their access frequency. Then it takes the higher frequency of the first records for further analysis so as to obtain their location which includes the host name and the block number, thu put the location information in the client's memory. The analysis period can be set according to the requirements of the system. The index in the client memory for the prefetching mechanism is shown in Figure 3.

MergeFile 1	Host Name	Block Number
MergeFile 2	Host Name	Block Number
MergeFile 3	Host Name	Block Number
MergeFile 4	Host Name	Block Number
MergeFile 5	Host Name	Block Number
. . .	. . .	. . .
MergeFile n	Host Name	Block Number

Figure 3: Index for Prefetching

The N and n are not fixed value, they can be modified according to the hardware configuration and the client's memory size.

3.1.4Reading Process

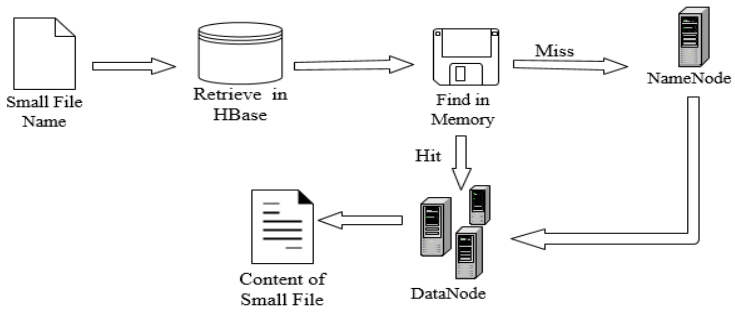

Figure 4: The Reading Process

When the system accesses a small file, the client firstly retrieves to HBase based on the small file name in order to acquire its corresponding merge file name, the size of the small file and the offset. Then, according to the merged file name, it finds the prefetch index table existed in memory whether there is a metadata for it. If hit, the reading request is sent to the specified host directly according to the host name that is retrieved in the prefetch index table and the block number. Combined with the small file size and offset obtained from the HBase table, the specified small file located in the merge file is available.

If it does not exist, it can only read according to the method of original HDFS. By sending a RPC request to the NameNode, we can request the metadata of the merge file. Then, it will send the read request to the DataNode and get the small file. The reading process is shown in Figure 4.

Based on the reading process above, we can draw that the client needs not send a request to the NameNode and get a response from it in the case of hitting, thereby improving the efficiency of reading.

3.2Enforcement Algorithm

For the merging files and establishing mapping index processes, the algorithm is as follows:

Algorithm 1:
Input: small files, T1(time threshold) Output: mergeFile, indexFile
1.mergeSmallFiles(){
2. IF(mergeFile.size()+nextSmallFile.size() >= 64M || T1 <= Duration){
3. upload mergeFile to HDFS;
4. put IndexFile to HBase;
5. empty mergeFile;
6. empty indexFile;
7. }ELSE{
8. mergeFile.add(Content of smallFile);
9. indexFile.add(Information of smallFile);
10. }

This algorithm will operate when the small files are uploaded, then compute the sum of the current temporary merge file and the next small file. If the sum is more than 64MB or the time of the temporary merge file has been in memory is longer than T1, then upload the merged file to HDFS and put the index to Hbase. Then the temporary merge file and the index file are emptied for receiving the next small files. Otherwise, the system will add the next small file to the merged file and add the index about the small file to the index file.

For the prefetching process, the algorithm is shown in Algorithm 2:

Algorithm 2:
Input: N audit logs Output: n index
1.prefetch(){
2. logs = auditLogs.fetch(N);
3. sortResult = logs. choose(n);
4. index = sortResult.analyse();
5. putIndexToMemory(index);
6. }

This algorithm is executed periodically. The input is the last N access records taken from the audit logs and sorted by the access frequency. And then, take the higher frequency of n records for further analysis, and requestto the NameNode for the metadata of the n files and put the host name and the block number in the client's memory.

4.Experiments and Discussions

4.1Experimental Environment

The experiments are carried on a Hadoop cluster consistingof four computers, which sets a NameNode and three DataNode. And the hardware configuration of the computer is Intel Core i7, dual-core 3.4GHz processor, DDR3 4GB memory and 120 GB hard drive. The software configuration of each computer is Ubuntu 14.04 operating system, jdk1.8.0, hadoop2.6.0,

zookeeper3.4.5, hbase1.0.0. In addition, the number of copy of the data block is three. In order to test the write speed, memory usage, and read speed of this scheme, we conducted three experiments respectively.

The experiments take 100000 small files as the data set, and the size of small files ranges from 1 KB to 50 KB. The formats of the small files are txt, doc, jpg. We randomly select 20000, 40000, 60000, 80000 and 100000 files from the file set in the experiment.

4.2Uploading Speed

In order to analyze the uploading speed, five sets of data are uploaded by the improved scheme and the original HDFS respectively, and then record the time and each group of the experiment is done three times. The speed takes the average of three times and the experimental result is shown in Figure 5.

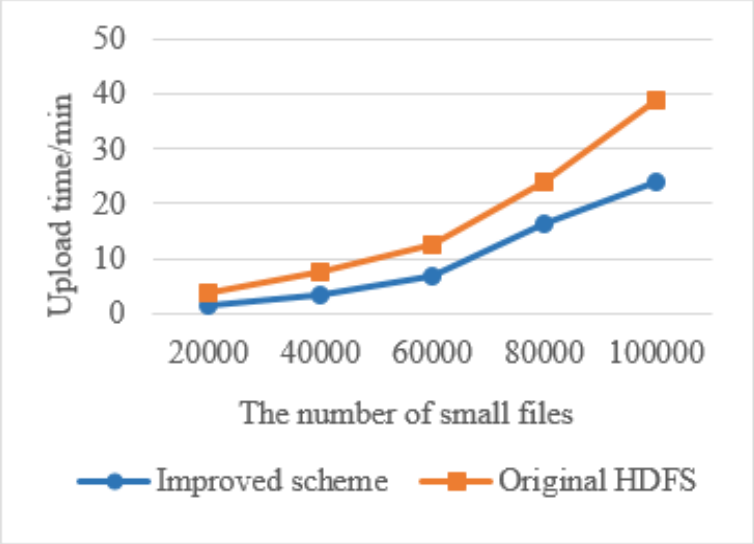

Figure 5: Time Comparison of Uploading Small Files

As is shown in Figure 5, when the number of files is 100000, the total writing time of original HDFS reaches 38.8min. For the improved scheme, the writing time is 23.8min, which is 61% of the original HDFS. And it proves that the proposed approach can effectively improve the efficiency of file writing. And we can also draw the conclusion that the effect of improved scheme will be more obvious with the increase in the number of small files.

4.3NameNode Memory Usage

In order to experiment the memory usage of the NameNode, five groups of data are uploaded by the improved scheme and the original HDFS respectively, then record the memory usage of NameNode and each group of the experiment is done three times. The results take the average of three times and the experimental results are shown in Figure 6.

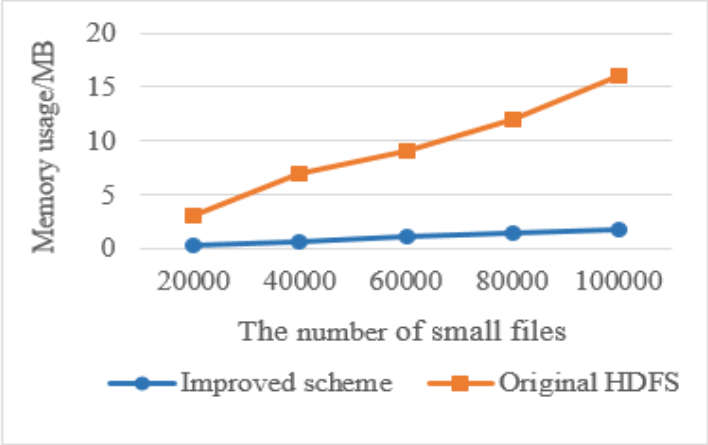

Figure 6: Comparison of NameNode Memory Usage

As verified by the results of the experiment testify, the improved scheme can greatly reduce the NameNode memory occupancy. When the number of files is 100000, 16MB of memory space is occupied through the original HDFS, while it is 2MB by the improved scheme, the memory usage reduced by 88% compared with the original HDFS. The great reason is that the small files are merged into large files to reduce the number of metadata files stored in NameNode.

4.4 Reading Speed

In order to analyze the reading speed, five groups of data are uploaded by the improved scheme and the original HDFS respectively. For the improved scheme, each group randomly generates 1000 access logs, then starts to read the small files according to the randomly generated file name and record the time it takes. Each group of the experiment is done for three times, and the results are achieved by the average time of such three experiment. The experimental results are shown in Figure 7. In the prefetching module, the values of N, n and T2 depend on the machine memory and specific application, etc. In this experiment, N values 1000, n values 100, analysis cycle T2 values 5s.

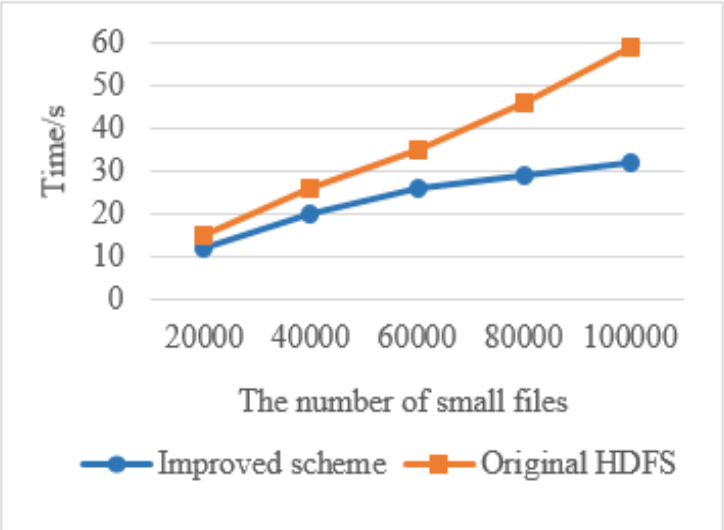

Figure 7: The Time Comparison of Reading Small Files

As the result of the improved scheme have added the prefetch mechanism, the speed of file access is accelerated obviously. When the number of files is 100000, the time spent on reading all these small files is 59s by the original HDFS, while the improved scheme is 32s, the reduced time is 46% by comparing with the original HDFS. And we can also find that the improved scheme shows a more obvious advantage with the increase of the small files.

5.Conclusion

This paper proposed a new scheme to improve the performance for storing the massive small files to HDFS, which mainly includes three processes: the merging small files, the establishing mapping index and the prefetching mechanism. For the merging small files, it significantly reduces the number of metadata files stored in the NameNode and decreases the overload of NameNode. The mapping index stored in the HBase is used to map to large files name based on the name of small files. The prefetching mechanism accelerates the speed of reading small files, and improves the performance of file access. Experimental results show that the speed of reading and writing small files has been improved considerably by our scheme based on the comparison between our scheme and the original HDFS. Hence, the overall performance of HDFS in dealing with massive small files has been improved greatly by this scheme. But there are still some shortcomings in this scheme, the relevance of small files and the abnormal situation during merging are not taken into accountwhich can be done in the further research.

References

[1]J. Dean, S. Ghemawat. *MapReduce: simplified data processing on large clusters*[J]. Communications of the ACM. 51(1), 107-113(2008)

[2]D. Borthakur. *The hadoop distributed file system: Architecture and design*[J]. Hadoop Project Website. 11(11), 1-11 (2007)

[3]Hadoop. *Hadoop Archives Guide*[EB/OL]. (2013-04-01)[2016-12-18]. http://hadoop.apache.org /docs / r1.2.1/Hadoop_archives. html

[4]Hadoop. *Sequence File Wiki*[EB/OL]. (2009-09-05)[2016-12-18]. https://wiki.apache.org/hadoop /SequenceFile

[5]C. Diem. *Combine File Input Format*[EB/OL]. (2013-09-22)[2016-12-18]. http://www.idryman.org /blog/2013/09/22/process-small-files-on-hadoop-using-ombinefileinputformat-1/

[6]S. Zhang, L. Miao, D. Zhang, Y. Wang. *A Strategy to Deal with Mass Small Files in HDFS*[C]. International Conference on Intelligent Human-Machine Systems and Cybernetics. IEEE ,NEW YORK. pp: 331-334(2014)

[7]A. Patel, M.A. Mehta. *A Novel Approach for Efficient Handling of Small Files in HDFS*[C]. IEEE International Advance Computing Conference. IEEE, NEW YORK. pp: 258-1262(2015)

[8]C. Vorapongkitipun, N. Nupairoj. *Improving performance of small-file accessing in Hadoop*[C]. 11th International Joint Conference on Computer Science and Software Engineering (JCSSE). IEEE,Washington DC. pp: 200-205(2014)

[9]B. Dong, J. Qiu, Q. Zheng, X. Zhong, J. Li, and Y. Li. *A novel approach to improving the efficiency of storing and accessing small files on hadoop: a case study by powerpoint files*[C]. 2010 IEEE International Conference on Services Computing (SCC). IEEE,USA. pp:65-72(2010)

[10]X. Liu, J. Han, Y. Zhong, C. Han, and X. He. *Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS*[C]. CLUSTER'09. IEEE International Conference on Cluster Computing and Workshops. IEEE. pp:1-8(2009)

[11]D. Beaver，S. Kumar，H.C. Li，J. Sobel，P. Vajgel. *Finding a needle in Haystack Facebook's photo storage*[C]. Usenix Symposium on Operating Systems Design and Implementation. ACM, NEW YORK. pp:47-60(2010)

[12]L. George. *HBase: the definitive guide*[M]. O'Reilly Media, America. pp:38-90(2011)