An Optimizational scheme of dealing with massive small files based on HDFS

Du Yongsheng
College of Computer
Jining University
Qufu JiNing Shandong, China

Abstract—**This paper delves the hadoop distributed file system-HDFS, and presents a new file combining, indexing and extracting scheme to alleviate the question that HDFS cannot perform well when facing with massive small files instead of the perfect efficiency of dealing with big files in order, which could reduce the usage of memory of Namenode and improve the I/O performance in HDFS.**

Keywords: massive small files, file merge, HDFS

I. INTRODUCTION

In view of the development of science and technology, data in various fields explodes, one single machine can never meet the requirement that dealing with massive amounts of data in a very efficient way, so the distributed way to store and process data turns to be of great significance. Hadoop [1], as an open source program, which funds by Apache, has been the most widely used distributed data processing framework. The distributed file system of Hadoop --HDFS [2], which has good fault tolerance and high throughput, can be deployed on inexpensive hardware and offers great data storage service which contributes to the suitability of dealing with applications that process extensive data sets. HDFS employs the master-slave scheme which is made up of one Namenode as the master node and several Datanodes as slave nodes.

Small file [3] is a comparative concept which refers to the file which size is smaller than the default size 64MB here. However, HDFS is first meant to process files with large scale in streaming way, but the scalability of HDFS will be badly constrained, and navigation between nodes will be very multiple, and then lead to the inefficiency of dealing with small files in Hadoop. However, the problem of the inefficiency exists even the HDFS is firstly designed, but the problem becomes much sever with the appearance of massive small files.

Aiming at this problem, this paper presents a new file combining indexing and extracting scheme in HDFS through combining small files, which gauges the size of one file first, if the size is less than 64MB, a file queue will be set up to list the small files and combines it into a big file block appropriately to reduce the number of small files and improve the overall performance of HDFS.

The rest of this paper is organized as follows. Section 2 introduces the related work, Section 3 explains the new file combining indexing and extracting scheme, and designed some experiments. Experimental results and analysis are presented in Section 4 and section 5 presents conclusions.

II. RELATED WORK

A Hadoop distributed file system --HDFS

HDFS is a perfect abstract distributed file model which uses the master-slaver architecture. A typical HDFS cluster is comprised of one Namenode and some Datanodes and one secondaryNamenode as well.

Namenode is the master node which plays a crucial role in storaging meta data in HDFS cluster, it saves and manages the data block mapping tables, the data node mapping tables and other key data structures.

Datanode, which keeps the ID mapping relationship and content of data bocks, takes the responsibility of communicating heartbeats between Namenode and Datanode regularly and communicates with other Datnodes as well to guarantee the consistency of operation.

B Extant problems and solutions of massive small files problem

Since Namenode needs to load all the files in FSImage to the memory after the HDFS cluster starts up, and each file occupies at least one file block, the total memory utilization becomes very low, the efficiency of data detecting and accessing turns to be very low as well.

Aiming at these problems, many scholars have carried out some researches and produced some results: the Hadoop group offered the HAR Files [4] scheme, which could archive several files into one file and allows for transparent access, and the SequenceFile [5] which operates files through the key/value form of which key refers to the name of binary file and value is the content of the same file; Shuo Zhang [6] et al. presented a novel small files reading strategy directed at different modes and different size, and proved that the novel strategy improved the efficiency of reading data by 75%; Chi Ziwen [7] et al. raised a file storage scheme based on key and value and improved the storage performance of distributed file system with massive small files.

Though all the improvements have made some contribution to save the memory of Namenode, these schemes consumed too much time, so this paper presents out a new scheme of small files processing and puts forward the pre-reading writing and extracting strategies using new structures.

III. SCHEME OF SMALL FILES PROCESSING

A Holistic approach of processing massive small files

Taking differences between the file size and the default size 64MB into consideration, HDFS pre-reserves a blank area. If size of the new file is smaller than the size of the pre-reserved blank area, the new file will be merged into the combining queue and waiting for being combined into file block in HDFS, otherwise, this new file will be put into next combining queue. A new serial number will be given to each new combined file, but two key questions exist here: the strategy of combining small files and the design of the mapping table in HDFS.

Based on the date, the writing time of the first file and time of the last file being created, structures of small and big file includes *Sf_Name Sf_Time* and *Sf_Data*, which represents name creating time and data content of small file, respectively, and *Sf_DelFlag* which refers to the deleting flag of small file. If the *Sf_DelFlag* is default value 0, the small file has not being deleted yet or the small file is already deleted. Structure of big file includes *Bf_Name Bf_Size* and *Bf_Data* which represents name size and data content of big file after combination.

Set the input *SmallFile[n]* and the output *BigFile[n]*, and assign the creating time of small file to t_1, then open a big file and read the value of *DelFlag* of one small file, and close the big file after the writing operation, take $t_1\#t_2$ as the name of the big file and return it. An indexing table, which contains the offset size of length and name of this small file, will be created and placed at the heading of the combined file. The file structure after combination is just like is shown in figure 1.

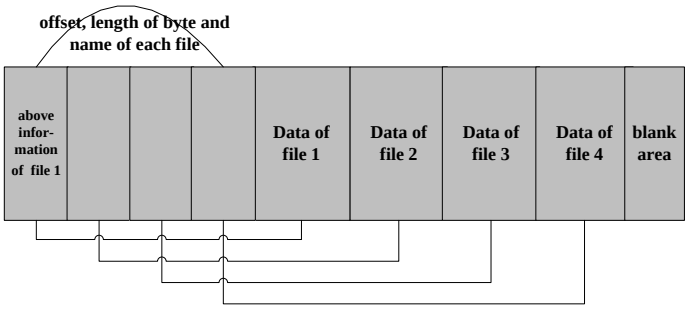

Figure 1 Structure of combined file

This makes getting the location of file and relevant information which can be used to achieve the aim function according to the name of the small file and then obtain the data of this small file accurately possible.

This paper takes ID of small files as the Rowkey and the column family sets up with ID of big file. The mapping table structure in HBase is like that showed in the following table 1:

▪ TABLE I Mapping Table Structure in HBase

Rowkey (ID of small file)	**TimeStamp**	**Colomn Family**		
		ID of big file	**Delete flag**	**Modification time**
small file 1	t_3	big file 1	0	141122146200
	t_6	big file 3	1	141122146200
small file 2	t_1	big file 1	1	141122146200
	t_3	big file 4	0	141122146700
.....	....	....	....	....
small file n	t_1	big file n	0	141122146900

B Scheme of pre-reading and extracting small files

Combining small files can only have a reduction on the footprint of Namenode, but really doesn't have anything to do with the increasement of the efficiency of reading small files, so the “Write Once Read Many (WORM) [8]“ pattern plays a very significant role in influencing the efficiency of reading small files. Under this condition, this paper designs and realizes a novel scheme of pre-reading meta data of one file to overcome the problem.

When the client trying to read one small file, meta data of the other file in the same combining file will be cached to the same client as well, then this client will never create new RPC requests. At the same time of reading files, the location of the small file data block can be

briskly got out, then the data will become available and being sent to the client at last.

Since there is no need to transfer the whole data block to the client, the network loads then could be significantly reduced.

IV. Experiments and Analysis

In this section, the comparative experimental results of the optimized method of dealing with massive small files based on HDFS are shown.

A Experimental Environment

The experimental platform is a Hadoop cluster composed of a single server as the Namenode and three clients as Datanodes which are both equipped with 2.10GHz CPU, 4G RAM and 320G hard disks. The operating system is Cent OS 6.5, the version of Hadoop is Hadoop-1.2.1, and all the showed experimental outcomes are the average results of the 3 runs.

B Results and Analysis

Since the memory usage of Namenode is closely bound up with the performance of HDFS, the consumption of memory of all three schemes – the traditional HDFS, the HAR scheme and the scheme of this paper will be presented firstly. Files in this experiment which include both images documentations and homepages etc and size of which are all less than 500KB are divided into four groups, number of files in each group is 20000,40000,60000,80000. Results are shown in figure 2 like bellow.

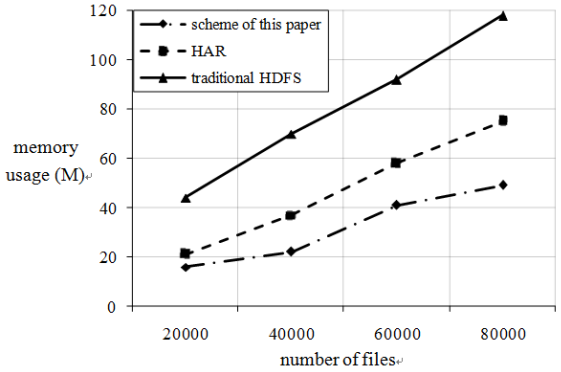

Figure 2 Comparison of memory usage of Namenode

This experimental result reveals that scheme presented in this paper takes much small footprint of Namenode and gains much better performance, such as the memory usage when the number of file is 40000, the traditional HDFS scheme takes up 70M memory, the HAR scheme holds onto 38M memory, but the scheme this paper presented occupies only 22M memory which reduces by about 49% compared with the traditional HDFS scheme and with much better performance than the HAR scheme, this phenomenon indicates that the new scheme presented here can resolve bottlenecks of the usage of memory of Namenode.

Then, take these three schemes into utilization and divide small files into four groups which number is 2000,4000,6000,8000, respectively. Time consumed by writing of the scheme presented in this paper includes both the time of combining files and the time consumed by writing files into HDFS after combination in the new scheme. Figure 3 illustrates the result.

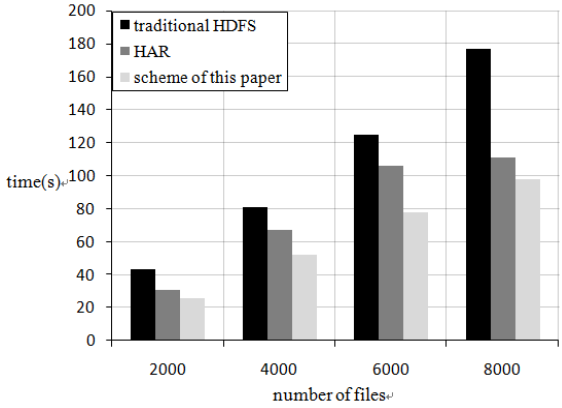

Figure 3 Comparison of writing time

The result, as is shown up, denotes that the new scheme consumes much less time than the other two schemes no matter dealing with how many small files even with the time taken by combining. That is to say, the new scheme obtains much better performance when writing.

Finally, take the experiment of reading using the same files with the writing experiment, and random extract 500 files from same groups, then record the time consumed by these three different schemes. The following figure 4 demonstrates the time consumed by each scheme.

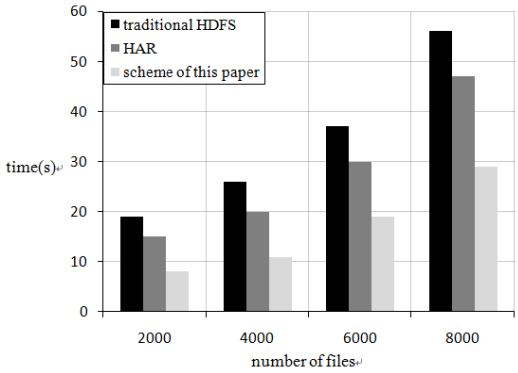

Figure 4 Comparison of reading time

It can be seen that time of reading files spent by the new scheme keeps less than both the traditional HDFS scheme and the HAR scheme. The traditional HDFS scheme consumes 26s and the HAR scheme takes 20s when the number of file is 4000, while the new scheme only consumes 11s, and the time-saving advantage will be much more significant with the increasement of the number of files.

Taken all together, the optimized method of dealing with massive small files based on HDFS could gain much better comprehensive performance than the traditional HDFS scheme and the HAR scheme especially as well when facing with much more small files.

V. Conclusion

The massive small file problem is a long standing problem which is a major performance hit to HDFS, this paper proposed an enhancement method for the massive small file problem with a new small file combining pre-reading and writing strategy and executes it on the most popular big data processing platform— Hadoop. The new scheme have gained good performance in dealing with massive small files, nevertheless, more attention should be invited to take much more node into comparison, at the same time, the data storage mechanism in HDFS should be perfected to improve data storage capabilities in future.

ACKNOWLEDGMENT

I would like to express my gratitude to all those who have helped me during the writing of this thesis. I gratefully acknowledge the help of the editor who have paied so much attention on reading this paper. Also, my gratitude also extends to my family who have been assisting, supporting and caring for me all of my life.

REFERENCES

[1] Hadoop, http://hadoop.apache.org/.

[2] Tom White: Hadoop: The Definitive Guide, edited by Tom White, Publications/O'Reilly Media Inc, USA (2010), in press.

[3] Parth Gohil and Bakul Panchal; "Efficient Ways to Improve the Performance of HDFS for Small Files" ; Computer Engineering and Intelligents Systems, Vol.5, pp. 45-49, November 2014.

[4] Bo Dong, Jie Qiu, Qinghua Zheng, Xiao Zhong, Jingwei Li and Ying Li; "A novel approach to improving the efficiency of storing and accessing small files on Hadoop: A case study by PowerPoint files"; Services Computing (SCC); Vol.7, pp. 65-72, July 2010.

[5] Andrzej Dziech, Andrzej Glowacz, Jacek Wszołek, Sebastian Ernst, Michał Pawłowski; "A distributed architecture for multimedia file storage"; Intelligent Tools for Building a Scientific Information; Vol.467, pp. 435-452, October 2013.

[6] Shuo Zhang, Li Miao, Dafang Zhang, Yuli Wang; "A Strategy to Deal with Mass Small Files in HDFS"; Intelligent Human-Machine Systems and Cybernetics (IHMSC); Vol.1, pp. 331-334, August 2014.

[7] Chi Ziwen, Zhang Feng, Du Zhenhong, Liu Renyi; "A distributed storage method of remote sensing data based on image blocks organization"; Journal of Zhejiang University (Science Edition); Vol.4, pp. 95-99, January 2014.

[8] P. Liu, T. P. Chen, Li X. D., Liu Z., Wong J. I., Liu, Y., Leong K. C.; "Realization of write-once-read-many-times memory device with O_2 plasma-treated indium gallium zinc oxide thin film" ; Applied Physics Letters; 104, pp. 37-43, January 2014.