Advances in Engineering Research, volume 140

2nd International Conference on Electrical, Control and Automation Engineering (ECAE 2017)

Research on Small File Processing Technology Based on HDFS

Rui Gu

[1]Full Address of First Author, China

***Abstract*—With the rapid development of the Internet and the rapid growth of Internet users, the Internet data is also a sharp expansion. The emergence of cloud computing is a good solution to the large data computing and storage problems, massive data storage and analysis has become a very popular research field.**

HDFS uses a single NameNode to manage the metadata of the entire system, and stores metadata in memory in order to improve access efficiency, but when the system stores a large number of small files, it generates a lot of metadata, occupies larger NameNode memory. In addition, a large number of small file access need to frequently send a request to the NameNode, resulting in the NameNode overload. In view of this problem, this paper analyzes some of the previous research and improvement programs, and on this basis to do a corresponding improvement. On the basis of the original distributed file system, an independent small file processing module was added. The small file processing module merged the small files, created the index of the file, and passed the file cache to HDFS for data processing.

Keywords—HDFS; cloud storage; small files; file merge; insert

I. INTRODUCTION

We live in an age of data explosion, and the amount of data increases exponentially over time. The emergence of cloud computing technology provides reliable technical support for solving, processing and storing huge amounts of data. The distributed file system, HDFS, which is part of Hadoop, an open source, distributed file system. Because distributed file system HDFS has many advantages, it has become the mainstream of data storage in distributed file system. HDFS has good reliability and scalability, and can run on low cost hardware clusters. The system stores documents through streaming data access patterns designed to store large files

There are lots of small files such as pictures, forms, fingerprints and so on in every field of the internet. Dealing with small files has become a hot topic of current research. And the HDFS is used to store large files, ignoring the handling of large amounts of small files. It has some performance problems in dealing with small files. This paper proposes a small file processing optimization method based on merging, indexing and file prefetching, which improves the storage efficiency of HDFS in dealing with small files.

II. RELATED BACKGROUND INTRODUCTION

It is difficult to store mass data in an operating system, then the data could be distributed to other system disks, but this is inconvenient for data management operations. Consequently, urgent needs to control many documents in a system. In this case, a distributed file system is created.

HDFS is the distributed file system of Hadoop. It consists of a master node and a number of slave nodes. And they transfer data through each other to command transmission, the specific structure shown in figure 1.

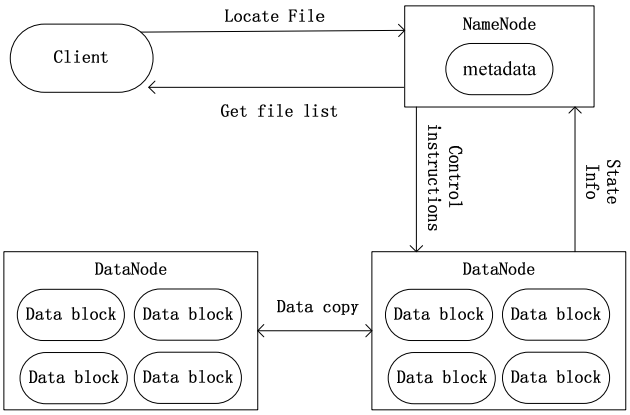

FIGURE I. HDFS SYSTEM ARCHITECTURE

The primary node and the slave node are the main functional components of the HDFS data store. NameNode is also known as a name node, and DataNode is called a data node: and NameNode has only one, and DataNode is uncertain, and can consist of many DataNode into a data pipeline.

A. Name Node

DataNode is the multiple independent node of the file system. It can really store data, through the client for the connection with the main node, and control of the recipient node, data storage and access. DataNode contains many file data blocks that carry file data. The file block is the most basic storage unit. The default file block size for HDFS is 64MB. DataNode is the place where data is stored in the distributed file system HDFS. Each DataNode will periodically send NameNode heartbeat information and each data block position information report to represent each data block operation is normal and can make NameNode timely and accurate grasp of each data node in DataNode data.

B. Data Interaction

Each node in HDFS needs data transmission and access to ensure the system running. Data interaction is first issued by

the client, and then carried out in each node, through certain protocols for data access requests.

C. Data Interaction

Since HDFS is based on the mechanism of one-time write multiple reads, it is more complex to write files relative to the read file. Name Node plays a key role in the process of writing data. Each client sends a request through the name node. The following is the procedure for writing files, as shown in figure 2.

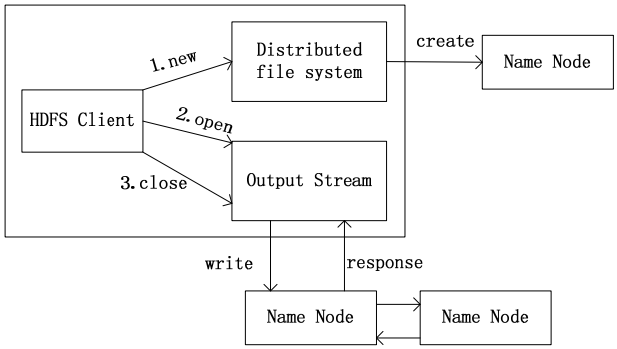

FIGURE II. THE PROCESS OF WRITING FILES IN HDFS

D. Data Interaction

The file read is also initiated by the client to the Name Node. The Name Node finds a data infor in the data block based on the requested content, then directly interaction with the Data Node. When the reading process is over, once the read is found to be incorrect, it is fed back to the Name Node for re reading. The read flow is shown in figure 3.

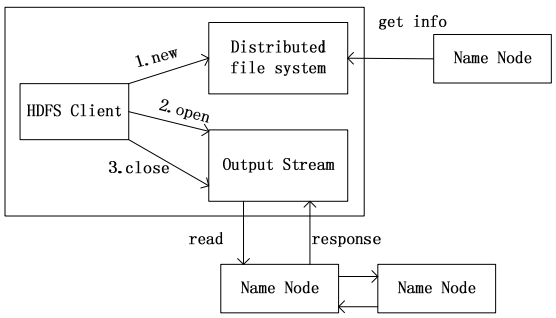

FIGURE III. THE PROCESS OF READING FILES IN HDFS

III. SOLUTION FOR SMALL FILE STORAGES

A. Problems with Small File Storage

The storage of small files has been a difficult problem in the Hadoop system. But what is a small document? To sum up, the reasons can be divided into two reasons. The first is a large number of data files, such as pictures and documents on internet; second small files are fragmented files, which can also be said to be a part of large files.

Hadoop was designed for streaming data access, which would be appropriate for handling large files. Small files generally refer to files smaller than the size of data blocks, whereas data blocks generally default to 64MB. Since each file corresponds to at least one data block and has metadata information, its metadata information is stored in the Name Node, the read and write of each file in the small file involves both the client and the name data transfer between Node and Data Node. It can be seen that a large number of small documents will inevitably affect the normal operation of the system, and even make the system paralyzed. Here are some of the low efficiency aspects of HDFS storing small files.

B. Memory Consumption of NodeName

HDFS is used as a data stream for reading and storage. The initial purpose of the design is to handle large files and is not suitable for storing large amounts of small files. In a HDFS cluster, the Name Node stores the file metadata and handle the operation request signal received by the client, which manages the file metadata. Since the Name Node has only one in the HDFS system, which means that its storage space is limited. In the document processing system, if the storage file for the majority of small files will obviously have a large number of small files, if most of the files that are stored are small files, there will obviously be a large number of small files, and if most of the files in the system are large files, the number of files will be much less than the smaller files that store the same size. Since each file corresponds to one metadata information stored in Name Node memory, it is clearly, that storing small files Name Node takes up more memory than storing large files.

In today's era of rapid Internet development, the number of documents increases with each passing day, while small files are part of the common files, and the number of them increases with time. If you store so many small files in the system, it will inevitably become a cumbersome system, resulting in a serious decline in performance of various aspects of the system. According to the research, the metadata information of a small file occupies about 150 bytes. If there are 1 million 500 thousand small file system memory may occupy the memory 450M, and if the number of small files up to 150 million requires at least 45G of memory, the system many small files does make the master node memory occupancy rate is too high.

C. Equations

All large file and small file data interaction, such as data access and data write through Name Node to achieve. With the same packet size, the number of large files is obviously larger. Because each operation involves frequent data interaction with the Name Node, which makes Name Node access data continuous, thus increasing the burden on the system, causing the system's data access can not be effectively completed on time. When you need to read data, Name Node finds the metadata information corresponding to the request file from its namespace, and returns Data Node block to the client. Because of the regular heartbeat detection between Data Node and Name Node, the Data Node will feedback data information on each node to the Name Node, Name Node will load the information into memory. There is a large amount of small files, each small file corresponds to a metadata information, so that the large part of memory is consumed in the data processing between nodes which makes the reaction speed of the HDFS system reduced.

IV. SOLUTIONS

A. Authors and Affiliations

(1) Merge a large number of small files in the system together. Since each file has a metadata, metadata needs to be stored in the Name Node. The transmission and access of files require frequent running of Name Nodes. If there are a large number of small files in the system, there will be a large amount of file metadata in the name of the corresponding file system, As a result, this will bring heavy burden to the system, slow down the system response and affect the ability of the system to process data. So the effective way to solve the problem is to reduce the number of files and merge files.

(2) Index to files that are grouped together. Although the files are grouped into blocks, reduced the number of total system files, however if you do not set up a quick search method, you will slow down the efficiency of system's access to data. As a result, we can efficient search the file by set up index, through which you can quickly locate the required data files

(3) Prefetching of small files. After merging and set index, small file processing efficiency has been greatly improved. Considering that the system consumes a lot of memory when merging the retrieved files, the cache prefetching of small files is added to optimize the overall system.

B. Scheme Design

Many of the existing solutions are specifically analyzed for specific problems, especially for specific system applications. To sum up, most small file processing methods have the steps of merging and indexing. Although these methods can improve the processing efficiency of files to some extent, they always bring some drawbacks to the system, and the system has a certain time delay when merging small files.

After summarizing these scenarios, this article creates a small file processing module outside the distributed system to process the data. We could merger documents, establishment of the file index in the small file processing module, then we could exchange data between processing module and system, Optimize the entire system by caching prefect of data. By creating an additional file processing module can run in this module without affecting the data in the system when the data is transmitted.

The paper is designed that the small file processing module working mechanism is to first confirm whether the data file is a small file, if it is a large file is uploaded to the HDFS system for data processing. If the file is small file, it will be entered into the small file processing module for data file merging, then the merged file will by build index tree by Map Reduce. Through the index matching small files and merged files, the establishment of small files and merged file mapping, and then in the small file processing module using preload mechanism to improve file access efficiency. According to the program design, we can draw the overall program structure shown in Figure 4.

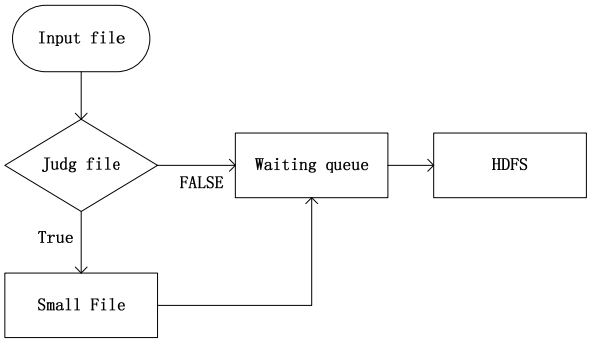

FIGURE IV. OVERALL DESIGN ARCHITECTURE

(1) Small file merge: Small file merge is that merge a lot of small files into a large file, after the merger of the file block is temporarily stored in the small file processing module.. File merging makes the total number of files reduced, the corresponding metadata will also be reduced, so you can reduce the system frequent data exchange, especially the Name Nodes and other parts, besides, after the merge file are uploaded to HDFS, there will be some metadata of the merged file remaining in the system.

(2) Small file merge: The design of this article by creating a two-level index for small files to reduce the disk lookup time, and these local index transmission placed in the small file processing module for the file prefetch to prepare, to more effectively improve the index to find the file effectiveness. When the merged files are uploaded to the cluster system, they are stored in the cluster, and the data blocks are automatically stored in the cluster's Data Node.In order to access small files in the merge file, we construct index for all files. The index of this program is similar to the HAR method, which create the two-level index. The index of this article is indexed the extension of the file and the name of the file. Through the secondary index to find first level index, through which we could position and read the small file. the retrieval process shown in Figure 5.

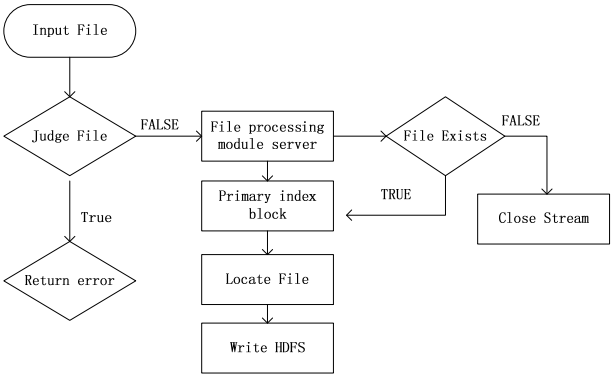

FIGURE V. THE PROCESS OF SMALL FILE RETRIEVAL

(3) Prefetch of small files: Merging small files only reduces the number of metadata in the Name Node, but does not change the system data access performance. Since the HDFS system accesses the data pattern is written once for multiple reads. As a result, read operation performance is particularly important. We can improve the speed of reading

operations by file cache and prefetched way. The original intention of set the file cache is that if there is a need to find the file directly in the small file processing module cache, So that you can greatly reduce the file data and the Name Node between the transmission link, thereby enhancing the efficiency of data processing. Index prefetch is through the small file processing module file index information for file prefetch. If the cache area has the index information, then through the index information for file retrieval and then directly read the small file, so you can avoid exchanging data with name node. If the index file is not prefetched, that is, the cache file does not request the index information of the file, In the cache area of the module, the metadata files that are left over from the name node is used as the initial file prefetch record.

V. CONCLUSIONS

This chapter first introduces the specific reasons for the low efficiency of the storage system dealing with small files, then analyzes several classical solutions and find out the advantages and disadvantages. Subsequently, in the conceptual design, the corresponding improvement has been made on the basis of inheriting the advantages of other solutions. The improved design of this paper is to add a small file processing module outside the original system, and then optimize the operation of the whole system by merging files, index and file prefetch in the small file processing module.

REFERENCES

[1] Implementing WebGIS on Hadoop:a case study of improving small file I/O performance on HDFS. LIU XUHUI,HAN JIZHONG,ZHONG YUNQIN, et al. CLUSTER''09:IEEE International Conference on Cluster Computing and Workshops . 2009

[2] A novel approach to improving the efficiency of storing and accessing small files on Hadoop:a case study by PowerPoint files. DONG B,QIU J,ZHENG Q,et al. SCC 2010:Proceedings of the 2010IEEE International Conference on Services Computing . 2010

[3] Hadoop:The definitive guide. WHITE T. . 2009

[4] The Small Files Problem. Tom White. http://max.book118.com the-smallfiles-problem/ . 2012

[5] Improving algorithm Apriori for data mining. Zhang Zhuo,Zhang Lu,Zhong Shao-Chun. 8th International Conference on Fuzzy Logic and Intelligent Technologies in Nuclear Science . 2008

[6] A view of cloud computing[J] . Michael Armbrust,Armando Fox,Rean Griffith,Anthony D. Joseph,Randy Katz,Andy Konwinski,Gunho Lee,David Patterson,Ariel Rabkin,Ion Stoica,Matei Zaharia. Communications of the ACM . 2010 (4).