Research on the Small Files Problem of Hadoop

Xiao Jun Liu[1, a *], Chong Peng [2,b] Zhi Chao Yu[3,c]

[1] School of Electronic & Information,Huanggang Normal University,Hubei Huanggang, China

[2] School of Electronic & Information,Huanggang Normal University,Hubei Huanggang, China

[3] School of Computer, Huanggang Normal University,Hubei Huanggang, China

[a] whutliuxiaojun@126.com, b 18623582@qq.com, c jsjyzc@hgnu.edu.cn

Keywords: Hadoop Distributed File System (HDFS); Small Files Problem; Hadoop Archives; Sequence files; RDBMS.

Abstract. Although Hadoop is widely used, its full potential is not yet put to use because of some issues, the small files problem being one of them. Firstly, the paper analyses the causes of the small files problem of Hadoop. Then, the current program to solve the small files problem are introduced, including Hadoop own programs and other application-specific solutions, and analyzes the advantages and disadvantages of various options. Finally, we present two research ideas, one is to use a combination of RDBMS and Hadoop; Another is to make the "Datanode" caching some metadata of the small files.

Introduction

Hadoop is economical and reliable, which makes it perfect to run data-intensive applications on commodity hardware. However, Hadoop distributed file system (HDFS) is designed to manage large files and suffers performance penalty while managing a large amount of small files. That is to say, small files are a big problem in Hadoop — or, at least, they are if the number of questions on the user list on this topic is anything to go by. In this paper we introduce the existing solutions against to the Small Files Problem of Hadoop, after analysising of their strengths and weaknesses, we propose future research directions.

Background

MapReduce[1]. MapReduce is a framework for processing highly distributable problems across huge datasets using a large number of computers (nodes), collectively referred to as a cluster or a grid. Computational processing can occur on data stored either in a filesystem (unstructured) or in a database (structured) [2].

In the Fig. 1, we can see the Map and Reduce tasks being assigned to the nodes by a master node, and the partitioned input given to the nodes assigned with Map tasks, which produces the intermediate values. The master node will be informed about the location of the intermediate values produced by each node. On acquiring this information, Master node will pass it to the nodes assigned with reduce tasks to finally perform the merging task, producing the output files.

Hadoop Distributed File System. Hadoop Distributed File System is the file system used by Hadoop. It is developed to support Hadoop in the data intensive distributed computations. In a cluster scenario where Hadoop is implemented, one node per cluster acts as the NameNode, which stores all the metadata of files. The application data will be stored in DataNodes [2-3].

If a client wants to perform a read operation, the request will be processed by the NameNode, and it will provide the location of data blocks which constitutes the file, and the client will perform the read operation from the closest DataNode. For a write operation, the NameNode will select a set of DataNodes to host the replicas of each block of the file and the client will write the file blocks into those DataNodes in a pipelined fashion.

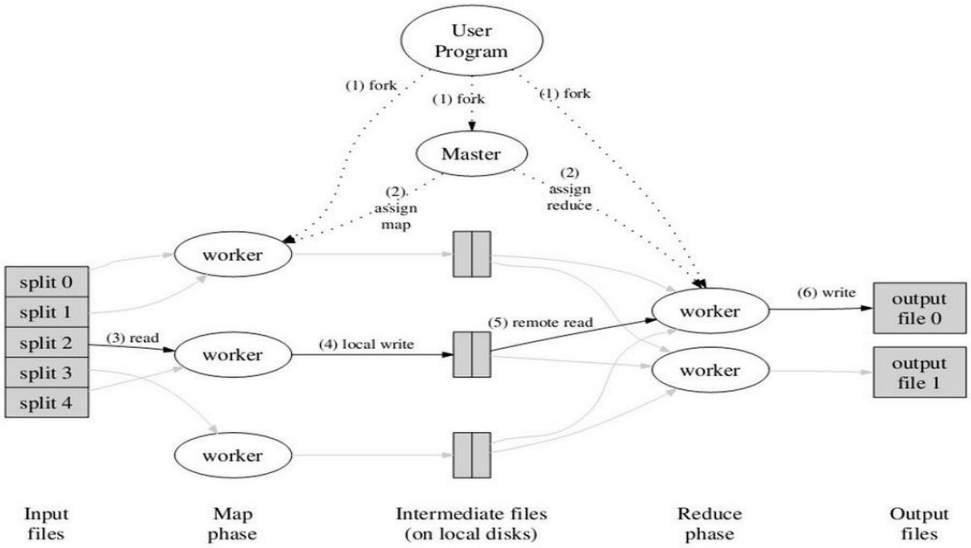

Fig. 1 MapReduce Execution overview[2]

In a cluster, during a DataNode start-up, it performs a handshake with NameNode which helps in maintaing data integrity. During handshake, the namespace ID and software version of the DataNode will be checked.Only a DataNode with the same namespace ID and supported software version will be allowed in the cluster[4][12].

A block report is sent periodically to the NameNode by each DataNode to provide information about the block replicas it holds which helps the NameNode in collecting information about the location of replicas of each block in a file, thus maintaining a consistent metadata. Also, DataNodes will send heartbeats every three seconds to acknowledge the DataNode about the availability of the node as well as the file block replicas it holds. If the NameNode did not receive the heartbeat of a DataNode within a particular time, say ten minutes, it will consider the DataNode as well as the block replicas it hosts as unavailable, and takes charge of creating new replicas of those blocks on other available DataNodes in the cluster [4].

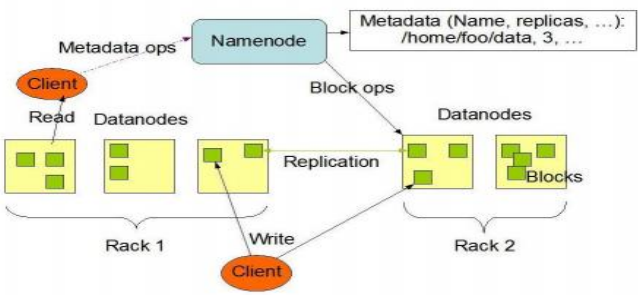

Fig. 2 HDFS architecture[2]

The NameNode allocates space for the metadata and balance the load among the DataNodes in its cluster using the information contained in the heartbeat. From the Fig. 2, we can get a pictorial view of HDFS architecture, as well as the read and write operations.

The Small Files Problem

Let us discuss the impact of this problem on the Hadoop Distributed File System and MapReduce, the two major components of Hadoop that we discussed before.

Problems with small files and HDFS.A small file is one which is significantly smaller than the HDFS block size (default 64MB). If you're storing small files, then you probably have lots of them (otherwise you wouldn't turn to Hadoop), and the problem is that HDFS can't handle lots of files[5].

Every file, directory and block in HDFS is represented as an object in the namenode's memory, each of which occupies 150 bytes, as a rule of thumb. So 10 million files, each using a block, would use about 3 gigabytes of memory. Scaling up much beyond this level is a problem with current hardware. Certainly a billion files is not feasible[3][5].

Furthermore, HDFS is not geared up to efficiently accessing small files: it is primarily designed for streaming access of large files [11]. Reading through small files normally causes lots of seeks and lots of hopping from datanode to datanode to retrieve each small file, all of which is an inefficient data access pattern.

Problems with small files and MapReduce. Map tasks usually process a block of input at a time. If the file is very small and there are a lot of them, then each map task processes very little input, and there are a lot more map tasks, each of which imposes extra bookkeeping overhead. Compare a 1GB file broken into 16 64MB blocks, and 10,000 or so 100KB files. The 10,000 files use one map each, and the job time can be tens or hundreds of times slower than the equivalent one with a single input file [2] [11].

There are a couple of features to help alleviate the bookkeeping overhead: task JVM reuse for running multiple map tasks in one JVM, thereby avoiding some JVM startup overhead (see the mapred.job.reuse.jvm.num.tasks property), and MultiFileInputSplit which can run more than one split per map[5] .

The own solutions of hadoop

HAR files. HAR files work by building a layered filesystem on top of HDFS. The Hadoop archives contains metadata and the data files. The metadata will be organized as index and master-index files, and data files will be stored as part-*.files. The name of the archive files and the location within the data files will be stored in the index file[6][11]. The modifications done to the file system for archiving is invisible to the user and yet, the increased system performance will be quite obvious to the user. Also, the number of files in HDFS has been reduced resulting in a better NameNode performance. Although files are archived, compression of files is not possible with this method. Reading through files in a HAR is no more efficient than reading through files in HDFS, and in fact may be slower since each HAR file access requires two index file reads as well as the data file read (see Fig. 3).

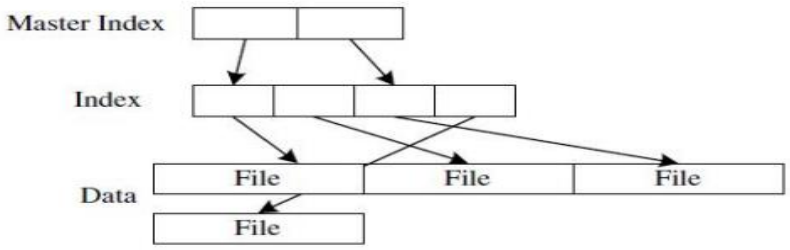

Fig. 3 Hadoop archive format [6]

Sequence Files. The idea here is that you use the filename as the key and the file contents as the value. This works very well in practice. Going back to the 10,000 100KB files, you can write a program to put them into a single SequenceFile, and then you can process them in a streaming fashion (directly or using MapReduce) operating on the SequenceFile. There are a couple of bonuses too[7]. SequenceFiles are splittable, so MapReduce can break them into chunks and operate on each chunk independently. They support compression as well, unlike HARs. Block compression is the best option in most cases, since it compresses blocks of several records (rather than per record) [11]. It can be slow to convert existing data into SequenceFiles. However, it is perfectly possible to create a collection of SequenceFiles in parallel. Going forward it's best to design your data pipeline to write the data at source direct into a SequenceFile, if possible, rather than writing to small files as an intermediate step.Unlike HAR files there is no way to list all the keys in a SequenceFile, short of

reading through the whole file. Conversion to sequence files might take time, and this method is mostly dependent on java i.e, it is not available in a cross-platform manner.

Furthermore, Paper [8] introduces a approach based on SequenceFile to improve storage efficiency of small files in the cloud storage systems that are on the basis of Hadoop distributed file system(HDFS). The approach uses the multi-attribute decision theory and the indices such as reading time, combining time, and saved memory size to obtain an optimal file merging scheme, so that the balance between computing time and memory space is achieved. A system load forecast algorithm is designed based on the analytic hierarchy process to predict the load of the system. SequenceFile is used to combine small files. Experimental results show that, without degrading the performance of storage system, the storage efficiency of small files is improved. This scheme is actually a kind of SequenceFile's optimization technology, it is the same with "SequenceFile" in essence.

Other solutions

While Hadoop Archives succeeded in grouping the small files, the read operation can still be slow as it requires reading the two index files and finally the data file for a single read. On the other hand, sequence files are ecient in data processing but it is platform dependent. And these solutions are required users to develop their own program.

Reference [9] and [10] were put forward their own solutions in view of the different application, in fact the thought of the two papers is similar: the basic idea is to combine small files into large ones to reduce the file number and build index for each file. in the original HDFS basis to add a small document processing module, when a file arrived, judge whether the file belongs to a small file, if it is, it gave it to the document processing module processing, otherwise store in HDSF directly.

Furthermore, In Paper [9], some novel features such as grouping neighboring files and reserving several latest version of data are considered to meet the characteristics of WebGIS access patterns.

Paper [10] introduces a novel approach to improve the efficiency of storing and accessing small files on HDFS. Characteristics of file correlations and access locality remaining among small files in the context of courseware are well considered for storing and accessing them. Firstly, all correlated small files of a PPT courseware are merged into a larger file to reduce the metadata burden on NameNode. Secondly, a two-level prefetching (That is an indexed file prefetching and data file prefetching) mechanism is introduced to improve the efficiency of accessing small files. Indexed file prefetching means that when users access to a file, the indexed file in the corresponding block has been added to the memory, so that users access to these files don't have to namenode interaction. Data files prefetching means that users visited a file, the block which the file in will be in all load to memory, so, if the user continue to visit other files, speed will be improved obviously.

Conclusion and Proposed Solution

Against small files problem, Hadoop provides several common solutions, including Hadoop Archive, Sequence files and CombineFileInputFormat, however, these schemes require the user to write programs according to their needs to achieve, and they also have their own flaws. Papers [9] and [10] which mentioned in Section four are proposed for specific applications solutions, they are not common technical solutions.

From this analysis we can see, the basic idea of existing programs is: small files merge into large files,and then stored on HDFS. However, the small files merge tool requires users to development by themselves, also need to establish an appropriate indexing mechanism, which is very difficult for users.

For small files merger, we think, one future research direction is to use RDBMS. We propose a comprehensive utilization of RDBMS and Hadoop cloud storage to their respective advantages, while avoiding their defects solution. The solution adopts two levels of storage mode, the front end uses lightweight database system to realize data access, including reading data and writing data, and the back end uses Hadoop cloud storage to store large database files. In order to avoid excessive amount

of data influence on the efficiency of the database system, database file whole replacement strategy is adopted to make database keep lightweight. Meanwhile, the merging of small files data is realized using RDBMS and large files are formatted and stored in the back end, avoiding small problem existed in Hadoop cloud storage.

In addition, from the above analysis, we can conclude that one of the main reason which leads small files problem in Hadoop is that the Namenode single point performance bottlenecks. So, we provide a different solution to deal with small files in HDFS that is to make the Datanode caching some metadata of the small file. The Datanode can process the request at the maximum probability when client post a request of small file through this way. It first find the data from the Datanode，if it can't find anything from that then it will find from the Namenode, this will reduce the amount of request processed by the Namenode greatly.

Acknowledgment

The project was supported by the College Students' Innovative Entrepreneurial Training Project in Hubei Province of China （201310514027）, and the Scientific Research Program of Huanggang Normal University (2014019103)

References and Notes

[1]. J. Dean and S. Ghemawat: MapReduce: simplified data processing on large clusters. Communications of the ACM, (2008), 51(1) p. 107-111

[2]. Mohandas, Neethu Thampi, Sabu M. Improving hadoop performance in handling small files. Communications in Computer and Information Science, v 193 CCIS, n PART 4, p 187-194, 2011, Proceedings of the First International Conference of Advances in Computing and Communications, ACC 2011,

[3]. HDFS Architecture (2010), http://hadoop.apache.org/common/docs/r0.20.1/hdfsdesign.html

[4]. Shvachko, K., Kuang, H., Radia, S., Chansler, R. The Hadoop Distributed File System. Proceedings of the 26th IEEE Symposium on Massive Storage Systems and Technologies (May 2010)

[5]. Information on http://www.cloudera.com/blog/2009/02/02/the-small-files-proble

[6]. Information on http://hadoop.apache.org/core/docs/r0.20.0/hadoop_archives.html

[7]. Information on http://hadoop.apache.org/common/docs/ current/api/org/apache/hadoop/io /SequenceFile.html

[8]. Yu Si, Gui Xiaolin, Huang Ruwei, Zhuang Wei. Improving the Storage Efficiency of Small Files in Cloud Storage, Journal of XI'AN JiaoTong University(Chinses), Vol 45 No.6 , Jun. 2011

[9]. Information on http://blog.yetitrails.com/2011/04/dealing-with-lots-of-small-files-in.html

[10]. Xuhui Liu, Jizhong Han, Yunqin Zhong, Chengde Han, Xubin He. Implementing WebGIS on Hadoop: A case study of improving small file I/O performance on HDFS. Proceedings of 2009 IEEE International Conference on Cluster Computing and Workshops. CLUSTER 2009: pp.1-8

[11]. Bo Dong, Jie Qiu, Qinghua Zheng, Xiao Zhong, Jingwei Li,. A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: A Case Study by PowerPoint Files. Proceedings of the 7th International Conference on Services Computing. SCC 2010. pp.65~72