International Journal of Application or Innovation in Engineering & Management (IJAIEM)
Web Site: www.ijaiem.org Email: editor@ijaiem.org
Volume 3, Issue 7, July 2014 **ISSN 2319 - 4847**

An Innovative Strategy for Improved Processing of Small Files in Hadoop

Priyanka Phakade[1], Dr. Suhas Raut[2]

[1]N.K. Orchid College of Engineering and Technology, Solapur

[2]N.K. Orchid College of Engineering and Technology, Solapur

ABSTRACT

Nowadays, the use of internet grows, so user wish to store data on cloud computing platform. Most of the time, user's data are small files. HDFS designed to process large volume of data. But it cannot handle large number of small files. In this paper, we have designed improved model for processing small files. In existing system, map tasks processes a block of input at a time. Map task produces intermediate output which is given to reducer. Reducer gives sort-merge output. Here, multiple map tasks & single reduce task is used for processing small file. In this approach, if there are large numbers of small files then each map task gets less input. In this way, performance of HDFS for processing lot of small files has been degraded. In proposed system, small files efficiently processed by HDFS. HDFS client requested to store small files in HDFS. NameNode permits to store small files in HDFS. When small files are submitted by HDFS client for processing, NameNode combine files into single split which becomes an input to map task. Intermediate output produced by map task is given to multiple reducers as an input. Reducer gives sorted merge output. As the number of map tasks has reduced, the processing time gets decreases.

Keywords: Hadoop Distributed File System, Hadoop, MapReduce, Small File Problem

1. INTRODUCTION

Hadoop is an open-source software framework developed for reliable, scalable, distributed computing and storage [10]. It is gaining increasing popularity for building big data ecosystem such as Cloud based on it [5]. Hadoop distributed file system (HDFS), which is inspired by Google file system [4], is composed of NameNode and DataNodes. NameNode manages file system namespace and regulates client accesses. DataNodes provide block storage, serve I/O requests from clients, and perform block operations upon instructions from NameNode. HDFS is a representative for Internet service file systems running on clusters [6], and is widely adopted to support lots of Internet applications as file systems. HDFS is designed for storing large files with streaming data access patterns [7], and stores small files inefficiently.
The Apache Hadoop framework is composed of the following modules:

- Hadoop Common - contains libraries and utilities needed by other Hadoop modules.
- Hadoop Distributed File System (HDFS) - a distributed file-system that stores data on the commodity machines, providing very high aggregate bandwidth across the cluster.
- Hadoop YARN - a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications.
- Hadoop MapReduce - a programming model for large scale data processing.

All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common and thus should be automatically handled in software by the framework. Apache Hadoop's MapReduce and HDFS components originally derived respectively from Google's MapReduce and Google File System (GFS) papers. Beyond HDFS, YARN and MapReduce, the entire Apache Hadoop "platform" is now commonly considered to consist of a number of related projects as well – Apache Pig, Apache Hive, Apache HBase, and others. For the end-users, though MapReduce Java code is common, any programming language can be used with "Hadoop Streaming" to implement the "map" and "reduce" parts of the user's program. Apache Pig, Apache Hive among other related projects expose higher level user interfaces like Pig latin and a SQL variant respectively. The Hadoop framework itself is mostly written in the Java programming language, with some native code in C and command line utilities written as shell-scripts[8].

2. RELATED WORK

Hadoop is an open-source Apache project. Yahoo! has developed and contributed to 80% of the core of Hadoop [3] from Yahoo! described the detailed design and implementation of HDFS. They realized that their assumption that applications would mainly create large files was flawed, and new classes of applications for HDFS would need to store a large number of smaller files. As there is only one NameNode in Hadoop and it keeps all the metadata in main memory, a large number of small files produce significant impact on the metadata performance of HDFS, and it appears to be the bottleneck for handling metadata requests of massive small files [2]. HDFS is designed to read/write large files, and there is no optimization for small files. Mismatch of accessing patterns will emerge if HDFS is used to read/write a large amount of small files directly (Liu et al., 2009). Moreover, HDFS ignores the optimization on the native storage resource, and leads to local disk access becoming a bottleneck. In addition, data prefetching is not employed to improve access performance for HDFS[1]. HAR is a general small file solution which archives small files into larger files [9].

3. Small File Problem

The Hadoop Distributed File System (HDFS) is a distributed file system. It is mainly designed for batch processing of large volume of data. The default block size of HDFS is 64MB. HDFS does not work well with lot of small files for the two reasons. First reason is that each block will hold a single file. Thus, if there are so many small files then there will have a lot of small blocks (smaller than the configured block size). Reading all these blocks, one by one means a lot of time will be spent with disk seeks. Another reason is the NameNode keeps track of each file and each block (about 150 bytes for each) and stores this data into memory. A large number of files will occupy more memory [11].

4. Proposed System

The Hadoop Distributed File System (HDFS) is designed to store and process large data sets. However, storing a large number of small files in HDFS is inefficient. The large number of small files increases the number of mappers, less input for each map task and overall processing time. Hadoop works better with a small number of large files than a large number of small files. Compare a 2 GB file broken into thirty two 64 MB blocks and 20,000 or so 100 KB files. The 20,000 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 & it uses 32 map tasks. The situation is alleviated by modifying InputFormat, which will be designed to work well with small files. FileInputFormat creates a split per file. InputFormat will be modified in such a way that many files will be combined into single split so that each mapper will have more to process. Therefore, as the each mapper will get sufficient input to process, the job completion time becomes less. In this way, this method minimizes the overall processing time. Fig.2 shows the proposed system for processing large number of small files. In first step, we will have to implement the middleware between the HDFS & HDFS Client. HDFS Client will request for connection to NameNode. NameNode will accept the connection. When HDFSClient will wish to store files in HDFS then HDFS client will want to take permission from NameNode. NameNode will grant or deny the permission. If NameNode does not allow to HDFS Client for storing files into HDFS then HDFS Client won’t store files into HDFS and the connection between the HDFS Client & NameNode will be closed. Otherwise HDFS Client will store the files into HDFS. Splits will be provided as an input to each DataNode. When a map task will be completed, it will generate the intermediate result which is given to reducer. After getting the input, the reducer will give the sort-merge output & finally the result will be stored on HDFS. Here, multiple reducers will be used for taking advantage of parallelism.

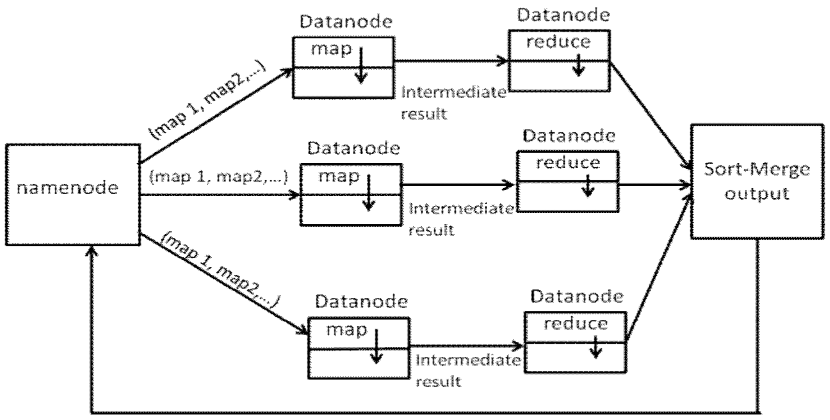

Figure 1: Workflow of proposed system

5. Expected Performance Improvement

In existing HDFS system, the Mapper maps input key/value pairs to a set of intermediate key/value pairs. Maps are the individual task that receives input records & results into intermediate records. The intermediate records do not need to be of the same type as the input records. A given input key/value pair may map to zero or many output pairs. The Hadoop MapReduce framework spawns one map task for each InputSplit generated by the InputFormat for the job. Reducer reduces a set of intermediate values which share a key to a smaller set of values. Multiple mappers & Single reducer is used for Word-Count application. As shown in result, hadoop job is executed on NameNode. For testing, 15,000 files were used for processing on Word-Count application. HDFS Client stores 5000 small files in HDFS System. In existing HDFS system, the word count application assigns 2546 map tasks & one reduce task to process those files. Processing time for 15000 test files was observed to be 1697 seconds.

In our proposed system performance be improved by modifying InputFormat class. InputFormat is responsible for splitting the input files. InputFormat will be modified in such a way that multiple files are combined into a single split. So, the map task will get more input to process, unlike existing system. As a result, the time required to process large number of small files will be minimized. In addition, multiple reducers will be used for taking advantage of parallelism. We expect at least 50% performance improvement with our proposed system.

6. CONCLUSION

We have studied performance behavior of HDFS for small file processing. Considerable time is needed to process a large number of small files. We are proposing innovative method for processing large number of small files. In this method, we will have to implement middleware between the HDFS Client & Hadoop Distributed File System. Also, we will have to modify InputFormat & task management of MapReduce framework. In existing system, the map tasks receives single block of input at a time. As there are so many small files to process, each map task requires less input for process. So it requires too much time for processing large number of small files. In proposed system, multiple files are combined into single split which is given as an input to map task. So, the number of map tasks has decreased by giving more input to each map task. In this way, the processing time required for many small files becomes less.

REFERENCES

[1.] Shafer J, Rixner S, Cox A. The hadoop distributed filesystem: balancing portability and performance. In: IEEE international symposium on performance analysis of systems & software (ISPASS). IEEE; 2010. p. 122–33.
[2.] Shvachko K, Kuang H, Radia S, Chansler R. The hadoop distributed file system. In: IEEE 26th symposium on mass storage systems and technologies (MSST). IEEE; 2010. p. 1–10
[3.] Mackey G, Sehrish S, Wang J. Improving metadata management for small files in hdfs. In: IEEE international conference on cluster computing and workshops, 2009. CLUSTER'09. IEEE; 2009. p. 1–4.
[4.] S. Ghemawat, H. Gobioff, and S. Leung, "The Google File System," Proc. of the 19th ACM Symp. on Operating System Principles, pp. 29–43, 2003.
[5.] Feng Wang, Jie Qiu, Jie Yang, Bo Dong, Xinhui Li, and Ying Li, "Hadoop high availability through metadata replication," Proc. of the First CIKM Workshop on Cloud Data Management, pp. 37-44, 2009.
[6.] W. Tantisiriroj, S. Patil, and G. Gibson, "Data-intensive file systems for internet services: A rose by any other name," Tech. Report CMU-PDL-08-114, Oct. 2008.
[7.] Tom White. Hadoop: The Definitive Guide. O'Reilly Media, Inc. June 2009.
[8.] Apache hadoop http://en.wikipedia.org/wiki/Apache_Hadoop
[9.] Hadoop archives, http://hadoop.apache.org/common/docs/current/hadoop_archives.html.
[10.] Hadoop official site, http://hadoop.apache.org/.
[11.] Solving the "Small Files Problem" in Apache Hadoop: Appending and Merging in HDFS http://pastiaro.wordpress.com/2013/06/05/solving-the-small-files-problem-in-apache-hadoop-appending-and-merging-in-hdfs/