Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 6, June 2014, pg.785 – 789

RESEARCH ARTICLE

Efficient Way for Handling Small Files using Extended HDFS

Kashmira P. Jayakar[1], Y.B.Gurav[2]

[1]Computer Department & PVPIT Bavdhan, Pune
Pune University, India

[2]Computer Department & PVPIT Bavdhan, Pune
Pune University, India

[1] kash.jayakar@gmail.com
[2] ybgurv@gmail.com

Abstract— Hadoop file system is for managing very large amount of files and high fault tolerant. Hadoop is widely used file system. Hadoopfile system is HDFS Hadoop Distributed File System. In HDFS, it is master-slave architecture, but in this architecture there is nothing shared between mater and slaves. Master is single server i.e. NameNode which stores metadata, in its own memory. As consequence, HDFS become problematic with number of small size files. But for storing and managing penalty number of small size imposes Burden on the NameNode, so it is difficult for NameNode to manage all DataNodes & burden on its memory. In HDFS, it does not provide any prefetching, combined file to improve the I/O performance. Extended Hadoop Distributed file system (EHDFS) improves storing and accessing efficiency of small file on file system, In this approach the files are stored in one file is known as combined file on Datanode or client. The ConstituentFileMap is most important in EHDFS for efficient management. Indexing mechanism is used to access individual files from combined file. To minimize load on NameNode and I/O performance improvement index prefetching is used. The footprint in NameNode is reduced then the file system becomes more efficient. EHDFS will minimize the time required for processing.

Keywords— Hadoop, NameNode, Datanode, HDFS, EHDFS, ConstituentFileMap

I. Introduction

The Hadoop framework transparently provides applications reliable, scalable, data motion. Hadoop includes Map/Reduce, where application is divided into many modules, each of which may be executed or reexecuted on any node in cluster. Hadoop is software that easily writes and run applications which process vast amount of data. Hadoop includes MapReduce which offline computing engine, HDFS Hadoop Distributed File System and HBase (pre-alpha) online data access. The Hadoop framework which provides the facility of running application on large clusters made up of commodity hardware. The key of hadoop distributed file system is create modules of the input data, scheduling the program's execution across several machines,

© 2014, IJCSMC All Rights Reserved

handling machine failures and manages inter machine communication.

Hadoop distributed File System that stores data on computer nodes, providing high aggregate bandwidth across cluster. The main design principal of Hadoop is automatic parallelization and distribution which is hidden from the end- users. Fault tolerance and automatic recovery, if node/task fails, it recover automatically. Clean and simple programming abstraction. HDFS follows write once and read many times pattern. It is Master/slave architecture; HDFS exposes a file system namespace and allows user data to be stored in file. HDFS cluster consist of a one NameNode, a master server that manages the file system namespace and regulates access to files by clients. It consists of number of Data nodes per node in cluster. Data node manages storage attached to the nodes that they run on.[7] In Datanode a file is split into one or more set of blocks. Data nodes performance various operations such as serves read, write request, performs block creation, block deletion and replication upon instruction from NameNode. The NameNode maintains the metadata on its own main memory to ensure fast access to client, on read and write requests.

The rest of this paper is organized as follows: Section II describes Existing System, Section III describes Problem in Existing System, Section IV explains Propose system, Section V concludes and Provides possible future directions.

II. Existing System

Hadoop is an open source framework for providing distributed storage and data processing capabilities to data intensive applications. It consists of two major components: Hadoop Distributed File System (HDFS), for distributed storage and MapReduce, for distributed computation.

A. Hadoop File System

Hadoop distributed file system is used in a situations where massive amounts of data need to be processed. HDFS has highly fault-tolerant property and it is deployed on low-cost hardware. HDFS also provide high throughput access to application data and it is suitable for the applications which have large data sets. The NameNode also executes file system namespace operations like opening file/directories, closing file/directories, renaming file/directories.[8] It also performs the mapping of blocks to DataNodes. The architecture if HDFS is shown in Figure 1.

The block size and replication factors are configure as per file, the file replicated for fault tolerance. At file creation time the replication factor is specified and can be changed later.

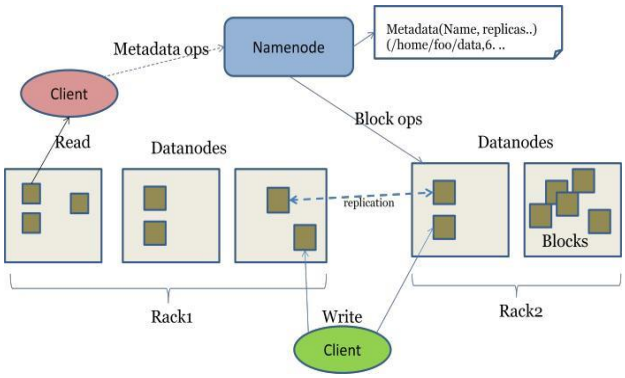

Fig 1. HDFS architecture

Block replication decisions are taken by NameNode. NameNode receives a heartbeat and a block report from each of the Datanodes in the cluster. A block report contains a list of all blocks on a DataNode. Receipt of a heartbeat implies that the DataNode is functioning properly.

III. Problem in Existing System

Client request to name node to access the metadata which stores in main memory of Namenode. When a file whose size is larger than block size is stored, the amount of metadata stored is justified by the size .When a large number of small files, the block size is greater than the file size, and the corresponding metadata stored is high.

© 2014, IJCSMC All Rights Reserved

Hadoop has a serious Small File Problem. It's widely known that Hadoop struggles to run MapReduce jobs that involve thousands of small files: Hadoop much prefers to crunch through tens or hundreds of files sized at or around the magic 128 megabytes.[8]

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.[10]

Every file, directory and block in HDFS is represented as an object in the NameNode's memory, each of which occupies 150 bytes. 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.

IV. Proposed System

The modified HDFS is nothing but the Extended High Distributed System (EHDFS).EHDS is used to improve the efficiency with handles small files. EHDFS include four operations i.e. File merging, File mapping, prefetching and file extraction shown in Fig 2.

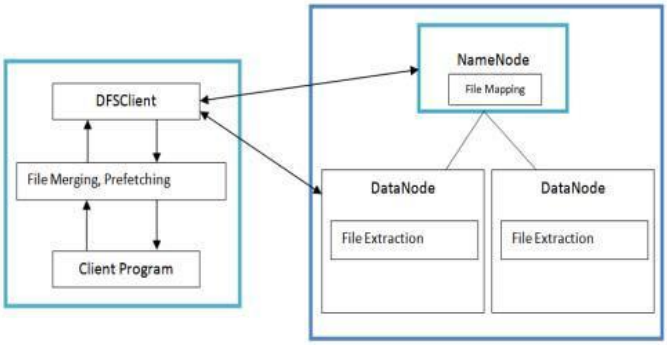

Fig 2 . Architecture of Extended-HDFS

A. *File Merging*

HDFS is the amount of memory used by the NameNode to manage these files. Differentiation between a small file and a large file does not in HDFS and hence stores the same amount of metadata regardless of the file size. In HDFS, NameNode maintains two types of metadata i.e. the file metadata and the block metadata.

The NameNode also provides an "index entry number". This "index entry number" specifies which entry in the index table stored at the beginning of the block corresponds to the requested small file, thereby avoiding a linear search. File metadata manages information about the file such as location of the file in the name space tree, name of the file, file Size, access time, modification time, file permissions, and ownership details.[11]

File metadata and block metadata maintained by NameNode is reduced by file merging technique for small files. File merging ensures that the NameNode maintains metadata only for the combined file and not for all the small files present in it. The names of the constituent files and their block information are maintained as part of a special data structure in the NameNode. [5]

At the beginning of block, file data and index table is placed The table contains an entry for each small file that is a part of the block. Every table entry is an (offset, length) pair. For the m^{th} files in the block, the m^{th} entry in the index table specifies the offset of the first byte of the small file from the beginning of block and the length of the small file in bytes. The structure of the produced block, named as extended block, is depicted in fig 3.

© 2014, IJCSMC All Rights Reserved

B. *File Mapping*

When user wants to read a small file from combined file, File mapping technique comes into play. File mapping is the process of mapping large number of small file names to the block of the combined file that contains this file, complete process is carry out at NameNode.

ConstituentFileMap contains 8 small files spread over 3 blocks. The first block contains the files f1 and f2. The second block holds four files from f3 to f6, while the last block stores the remaining two files. Along with the name of the file, we maintain information about ordering of files in the block. The file names are hashed into the constituent file map.[7] This is done using the "index entry number" field. Fig 4 shows the ConstituentFileMap data structure for a combined file named temp.

The user has to explicitly. Specify the name small file while initiating the read operation of the combined file and the. The location of desired small file is obtained, when a request is sent to NameNode, along with two file names. For each combined file, NameNode maintains a data structure called as Constituent FileMap. It contains a mapping between a small file name and the logical block number of the combined file that holds this small file.[2]

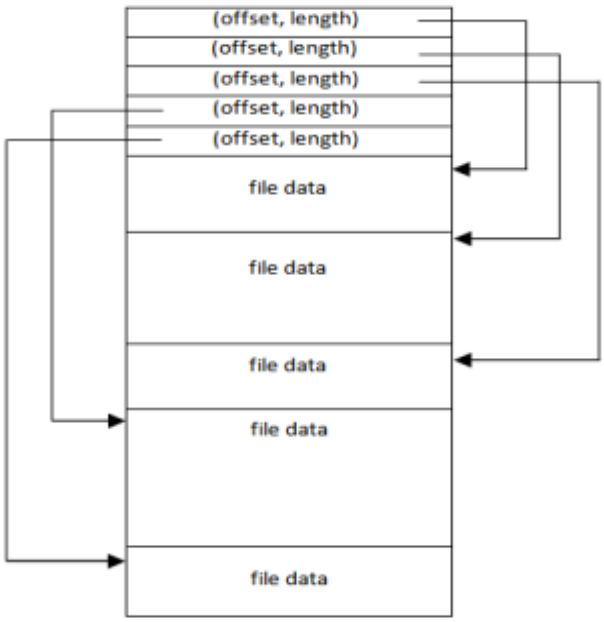

Fig 3.Extended block structure

C. *Prefetching*

ConstituentFileMap contains 8 small files spread over 3 blocks. The first block contains the files f1 and f2. The second block holds four files from f3 to f6, while the last block stores the remaining two files. The file names are hashed into the constituent file map. It is necessary to maintain information about ordering of files in the block. This is done using the "index entry number" field. Fig 4 shows the ConstituentFileMap data structure for a combined file named temp.[9]

© 2014, IJCSMC All Rights Reserved

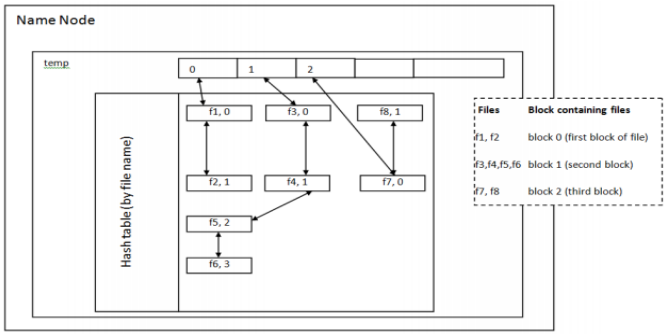

Fig 4. Constituent File Map structure

D. *File Extraction*

The process of Extracting a desired file contents from a block is called file extraction. DataNode performs the file extraction operation. File extraction operation not only reduce the load on the network as the entire block but also data is not sent back to the client. To obtain a the desired file, client specifies both the name of small file and name of combined file.[2] The obtained "index entry number" is then sent to the DataNode that has the block. The entry in index table contains the offset of the file data from the beginning of the block and also the length of the file data. The DataNode then uses this value and seeks to the desired entry in the index table placed at the beginning of the block. The DataNode then seeks to the desired offset and reads the requested file data and sends it to the client.[8]

V. Conclusion

HDFS was designed only to store large files. When large number of small files is stored in HDFS, I/O performance becomes the bottleneck. So metadata footprint in NameNodes main memory increases rapidly. The proposed EHDFS has been design to merge a large number of small files into a single combined file and it also provides a framework for prefetch metadata for a specified number of files.

EHDFS can be improves the efficiency of access to small files and reduces the metadata footprint in Name Node's main memory, corresponding to the small files. System performance lost is in proportion to number of node failed in HDFS, to avoid this problem we implement replication in Extended HDFS, so ultimately it will reduce system performance lost.[8]

References

[1] "Apache Hadoop". http://hadoop.apache.org/, 2009

[2] B. Dong, J. Qiu, Q. Zheng, X. Zhong, J. Li, Y. Li. "A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: A Case Study by PowerPoint Files". In Proceedings of IEEE International Conference on Services Computing, Miami, Florida, USA, July 2010, pp. 65 72

[3] Chuck Lam, Hadoop In Action, 1st ed. Dec. 2010, pp. 8.

[4] http://wiki.apache.org/hadoop/

[5] "HDFS Arhitecture Guide". http://hadoop.apache.org/common/docs/current/hdfs design.html, 2009.

[6] Jason Venner, Pro Hadoop, 1st ed. Apress, Jun. 2009, pp. 4 17

[7] S. Ghemawat, H. Gobioff, S. Leung. "The Google File System". In Proceedings of ACM Symposium on Operating Systems Principles, Lake George, NY, October 2003, pp. 29 43.

[8] Tom White, "The Small Files Problem". http://www.cloudera.com/blog/2009/02/the small files problem, 2009

[9] http://blog.cloudera.com/blog/2009/02/the-small-files-problem/

[10] http://www.idryman.org/blog/2013/09/22/process-small-files-on-hadoop-using-combinefileinputformat-1/

[11] http://amilaparanawithana.blogspot.in/2012/06/small-file-problem-in-hadoop.html

© 2014, IJCSMC All Rights Reserved