ISSN: 2321-7782 (Online)
Volume 2, Issue 8, August 2014
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Survey Paper / Case Study
Available online at: www.ijarcsms.com

Managing Small Size Files through Indexing in Extended Hadoop File System

K. P. Jayakar[1]
Department & PVPIT Bavdhan
University of Pune
Pune – India

Y. B. Gurav[2]
Computer Department & PVPIT Bavdhan
University of Pune
Pune – India

Abstract: HDFS (Hadoop Distributed File System) is a distributed user level file system which stores, processes, retrieves and manages data in a Hadoop cluster. HDFS infrastructure that Hadoop provides, include a dedicated master node called Name Node which contains a job tracker, stores meta-data,controls the overall distributed process execution by checking out whether all name nodes are functioning properly through periodic heart beats. It also contains many other nodes called Data Node which contains a task tracker, stores applications data. The Ethernet network connects all nodes. HDFS is implemented in Java and it is platform independent. Files in HDFS are split into blocks and each block is stored as an independent file in the local file system of Data Nodes. Each block of a HDFS file is replicated at least three times inmultiple Data Nodes. Through replication of application data, provides data durability. 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 an efficient management. To access individual file from combined file Indexing mechanism is use. 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 minimize the time period required for processing.

Keywords: Hadoop, NameNode, Datanode, HDFS, EHDFS, ConstituentFileMap.

I. Introduction

Hadoop is a framework written in Java for running applications on large clusters of commodity hardware and incorporates features similar to those of the Google File System (GFS) and of the MapReduce computing paradigm. Hadoop's HDFS is a highly fault-tolerant distributed file system and, like Hadoop in general, designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications that have large data sets

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

HDFS has a master/slave architecture. An HDFS cluster consists of a single NameNode, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of DataNodes, usually one per node in the cluster, which manage storage attached to the nodes that they run on. HDFS exposes a file system namespace and allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode.

The existence of a single NameNode in a cluster greatly simplifies the architecture of the system. The NameNode is the arbitrator and repository for all HDFS metadata. The system is designed in such a way that user data never flows through the NameNode.

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 Results, Section VI 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 NameNode and DataNode are pieces of software designed to run on commodity machines. These machines typically run a GNU/Linux operating system (OS). HDFS is built using the Java language; any machine that supports Java can run the NameNode or the DataNode software. Usage of the highly portable Java language means that HDFS can be deployed on a wide range of machines. A typical deployment has a dedicated machine that runs only the NameNode software.

HDFS was designed to be a scalable, fault-tolerant, distributed storage system that works closely with MapReduce. HDFS will "just work" under a variety of physical and systemic circumstances. By distributing storage and computation across many servers, the combined storage resource can grow with demand while remaining economical at every size.

Minimal data motion. MapReduce moves compute processes to the data on HDFS and not the other way around. Processing tasks can occur on the physical node where the data resides. This significantly reduces the network I/O patterns and keeps most of the I/O on the local disk or within the same rack and provides very high aggregate read/write bandwidth. Rack awareness allows consideration of a node's physical location, when allocating storage and scheduling tasks. Utilities diagnose the health of the files system and can rebalance the data on different nodes. Rollback allows system operators to bring back the previous version of HDFS after an upgrade, in case of human or system errors. Highly operable. Hadoop handles different types of cluster that might otherwise require operator intervention. This design allows a single operator to maintain a cluster of 1000s of nodes Standby NameNode provides redundancy and supports high availability.

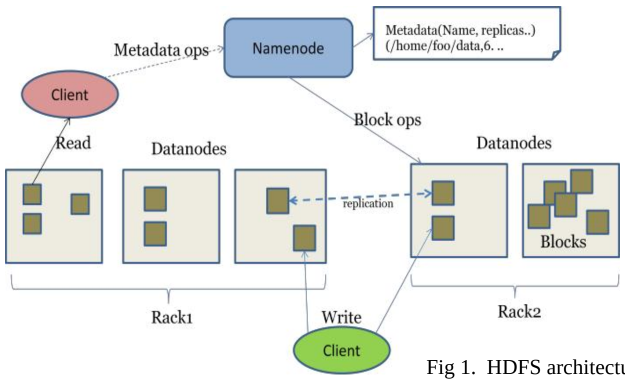


Fig 1. HDFS architecture

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, the corresponding metadata stored is high.[12]

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, 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.

Small files run into thousands and cause many issues in production :--

1) Each file reference accounts to ~150 bytes of memory in the Name node. Thats about 1GB for every million blocks. As the number of files on the HDFS grows, the size of the fsimage grows. Hence, the name node process runs into the danger of slowness.

2) While running map/reduce jobs. The loading of thousands of small files cause a lot of time, specifically, if your small files are in snappy compressed form.

3) The number of blocks also increases as the number of small files increase. This causes, increase number of map tasks. Though increase in number of map tasks sometimes increase in parallelism and hence, job efficiency, but, too many mappers will cause too much load on the cluster causing slowness of job execution.

IV. Propose 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.

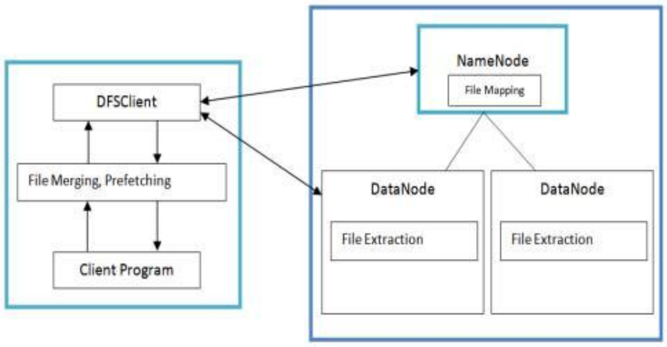


Fig 2. Architecture of Extended-HDFS

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.[13]

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.

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]

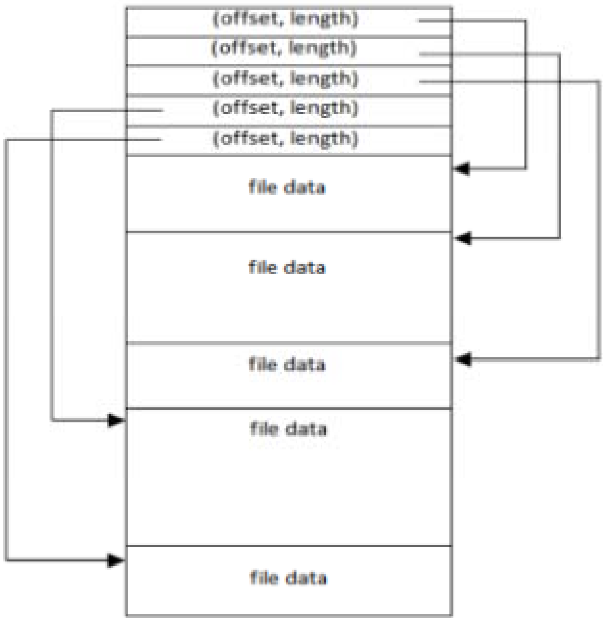


Fig 3.Extended block structure

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]

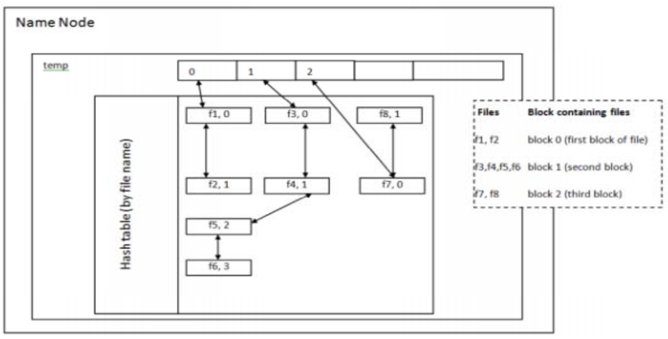


Fig 4. Constituent File Map structure

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. Results

A. *HDFS*

In Hadoop, map-reduce is most important part. Fig 5 shows the output of the HDFS system. At the start the user had given 2227 files as an input. All files are different type. To process all different type of files it needs 1 hour 30 mins. Firstly all 2227 files are map and then reduce.

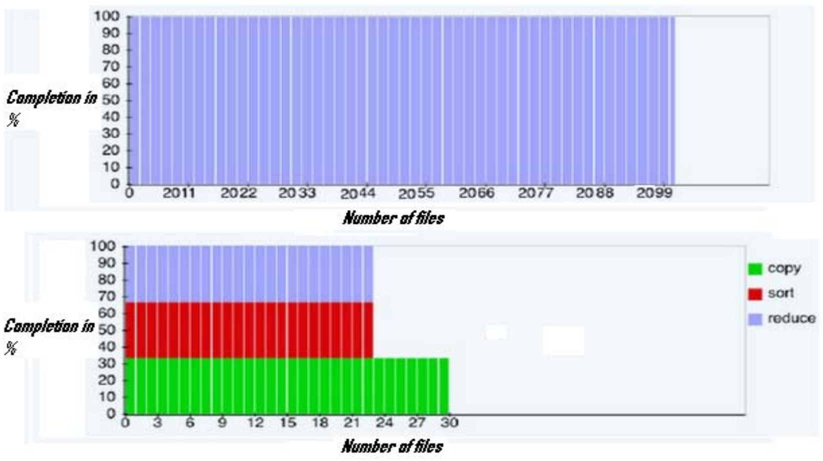


Fig.5 Output of HDFS

B. EHDFS

An extended Hadoop distributed file system is very reliable, and easy to use as compare to HDFS. EHDFS saves processing time compare with HDFS. In minimum time period EHDFS produces the expected output. It also minimise the burden on main NameNode.EHDFS is able to reduce the metadata footprint on Name Node's main memory. It also improves efficiency storing and accessing large of number small files.

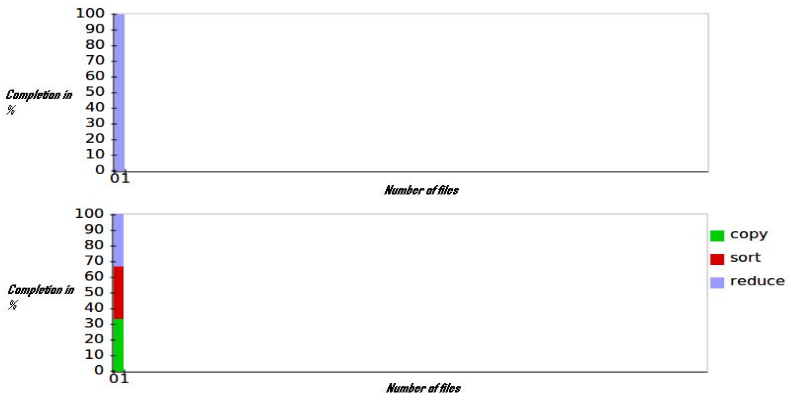
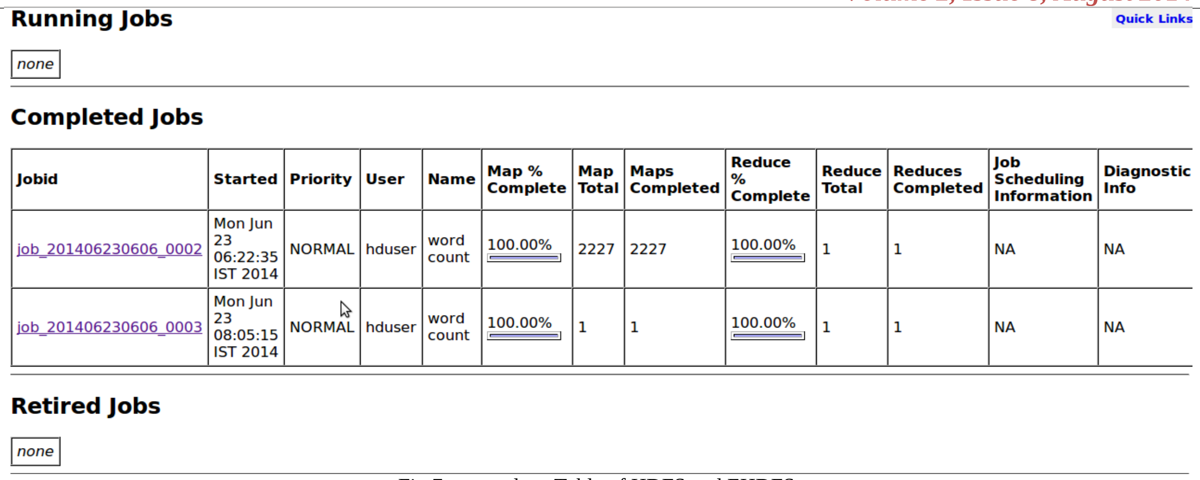


Fig 6. Output of the EHDFS

C. Comparison Of Map-Reduce Operation In HDFS And EHDFS

It describes the current usage Live/retired Nodes in HDFS and EHDFS in fig 7. Job_201406230606_0002 is the HDFS job, in this job it takes 2227 files as input and performs the map-reduce operation. After performing map-reduce only one map file is created. Job_201406230606_0003 created after performing EHDFS operation. In EHDFS the only one map file is given as input and EHDFS operations are performed. The time required to perform the EHDFS operation is very low as compared to HDFS and memory usage become low.

Running jobs shows the currently running files which are completing its own task if currently no job is running ,then it is indicated by none label. Completed jobs shows jobs whose task is completed with the attributes like priority specifies the priority of the job such as NORMAL, HIGH, LOW. User indicate the user name, name is the name of the command. Map % complete and Reduce % complete refers to percentage of the map operation completed, percentage of the reduce operation completed .Map Total shows the number of the files were processed. Retired jobs shows jobs whose functionality is not in use.



Running Jobs

none

Completed Jobs

Jobid	Started	Priority	User	Name	Map % Complete	Map Total	Maps Completed	Reduce % Complete	Reduce Total	Reduces Completed	Job Scheduling Information	Diagnostic Info
job_201406230606_0002	Mon Jun 23 06:22:35 IST 2014	NORMAL	hduser	word count	100.00%	2227	2227	100.00%	1	1	NA	NA
job_201406230606_0003	Mon Jun 23 08:05:15 IST 2014	NORMAL	hduser	word count	100.00%	1	1	100.00%	1	1	NA	NA

Retired Jobs

none

Fig.7 map-reduce Table of HDFS and EHDFS

VI. Conclusion

In EHDFS, Metadata footprint in name node's main memory is reduced and improvement in accessing and storing files. Improvement in accessing and storing proves that the time required for processing is also reduced. It minimise Number of node failure in EHDFS, so ultimately it will reduce system performance lost. Burden on name node's main memory and time required for processing is reduces using EHDFS.

Input /Output performance improves, and footprint in NameNode memory is also reduces. It is positive impact on the large number of small files. EHDFS generates the results in minimum time duration, and produced result indicated best result compare to HDFS. As for future work, this solution can be improved further to provide a more advanced prefetching framework Appendoperation can be provided by EHDFS, to add files into an existing combined file.

References

1. Jason Venner, Pro Hadoop, 1st ed. Apress, Jun. 2009, pp. 4 17
2. Tom White, "The Small Files Problem". http://www.cloudera.com/blog/2009/02/the small files problem, 2009
3. Chuck Lam, Hadoop In Action, 1st ed. Dec. 2010, pp. 8.
4. 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.
5. "HDFS Arhitecture Guide". http://hadoop.apache.org/common/docs/current/hdfs design.html, 2009.
6. "Apache Hadoop". http://hadoop.apache.org/, 2009
7. http://wiki.apache.org/hadoop/
8. 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
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
12. hvachko, K.; Hairong Kuang ; Radia, S. ; Chansler, R. " The HadoopDistributed File System". Published in: IEEE 26th Symposium on MassStorage Systems and Technologies (MSST), Incline Village, NV, May2010. Page(s): 1 - 10. E-ISBN: 978-1-4244-7153-9. Print ISBN: 978-1-4244-7152-2. INSPEC Accession Number: 11536653. D.O.I:10.1109/MSST.2010.5496972.
13. http://www.orzota.com/single-node-hadoop-tutorial/
14. http://www.orzota.com/multi-node-hadoop-tutorial/
15. http://hadoop.apache.org/docs/r0.18.0/hdfs_design.pdf