Computer Engineering and Intelligent Systems
ISSN 2222-1719 (Paper) ISSN 2222-2863 (Online)
Vol.5, No.1, 2014

Efficient Ways to Improve the Performance of HDFS for Small Files

Parth Gohil
Department of Computer Science and Engineering
Government College of Engineering,Modasa,Gujarat.
Tel: +91 9662123479 Email: er.parthgohil@gmail.com

Prof. Bakul Panchal
Department of Information Technology
L.D College of Engineering,Ahmedabad,Gujarat.
Tel: +91 9426436891 Email: bakul.panchal@gmail.com

ABSTRACT:-
Hadoop , an open-source implementation of MapReduce dealing with big data is widely used for short jobs that require low response time. Facebook, Yahoo, Google etc. makes use of Hadoop to process more than 15 terabytes of new data per day. MapReduce gathers the results across the multiple nodes and return a single result or set. The fault tolerance is offered by MapReduce platform and is entirely transparent to the programmers. HDFS (Hadoop Distributed File System), is a single master and multiple slave frameworks. It is one of the core component of Hadoop and it does not perform well for small files as huge numbers of small files pose a heavy burden on NameNode of HDFS and decreasing the performance of HDFS. HDFS is a distributed file system which can process large amounts of data. It is designed to handle large files and suffers performance penalty while dealing with large number of small files. This paper introduces about HDFS, small file problems and ways to deal with it.
Keywords**:** *Hadoop; Hadoop Distributed File System; MapReduce; small files*

1. INTRODUCTION

As data and work grow, it takes a longer time to produce results. To produce the result in timely manner, one should start thinking big. A certain way is to carried out to spread the work across many computers i.e. one needs to scale out. Data is classified as structured and unstructured data. Structured data uses a fixed layout (in form of tables and rows) and unstructured data uses the data in the form of logs (no tables). MapReduce is a programming model which is designed for processing large volumes of data in parallel[2]. It does so by dividing the work into a set of independent tasks. MapReduce programs transform lists of input data elements into lists of output data elements. The MapReduce data elements are immutable, i.e. they cannot be updated. The policy is to place the file into HDFS once and can read the file 'n' number of times.

If in a mapping task you change an input (key, value) pair, it does not get reflected back in the input files; communication occurs only by generating new output (key, value) pairs which are then forwarded by the Hadoop system into the next phase of execution. Hadoop finds difficulty in running MapReduce jobs which involve thousands of small files.

In this paper, we focus on the different efficient ways in which small files are handled in HDFS to improve performance of it.

2. HADOOP

Big data is used to handle a large amount of data and it considers unstructured data into account. Facebook is one of the best examples of big data. It is because it processes more than 15 terabytes of new data per day[2]. Say there is a machine having four I/O channels each with a throughput of 100 MB/sec and requires three hours to read a 4 TB data set. With Hadoop, the same data set will be divided into smaller blocks (typically 64 MB) and are distributed among many machines in the cluster via the Hadoop Distributed File System (HDFS)[2]. The cluster machines have the capability of reading the data set in parallel and providing a much higher throughput due to the degree of replication. A cluster of commodity machines is much cheaper than one high-end server.

The main challenge is to read-write the data or analyse the data. For this one requires a system that supports Distributed File System (DFS). It is a method of storing and accessing files based in client/server architecture. In it one or more central servers store files that can be accessed, with proper authorization rights, by any number of remote clients in the network.

Hadoop provides a framework for supporting this and is written in Java Programming Language. Apache Hadoop[1] is a software framework for distributed processing of large datasets across clusters of commodity

computers using single programming model. Large datasets means that Terabytes or Petabytes of data is stored. Commodity computers are that there is no need of high quality machine and a Single programming model i.e. MapReduce is used. In MapReduce the application is divided into many small fragments of work, each of which can execute or re-execute on any node in the cluster. It also provides a distributed file system (DFS) that stores the data on the compute nodes, providing very high aggregate bandwidth across the cluster. DFS is designed so that node failures are automatically handled by the framework. Hadoop streaming is a utility that comes with the Hadoop distribution which allows writing map/reduce code in any language one wants to. This is a key feature in making Hadoop more acceptable and attractive to use.

Companies using Hadoop are Yahoo, Google, Facebook, Amazon, IBM and Indian Adhar card System.

3. HDFS ARCHITECTURE

In this architecture, there are two main components of Hadoop namely (i) Hadoop Distributed File System (HDFS) which is used for storage purpose (ii) Execution Engine i.e. MapReduce for processing.

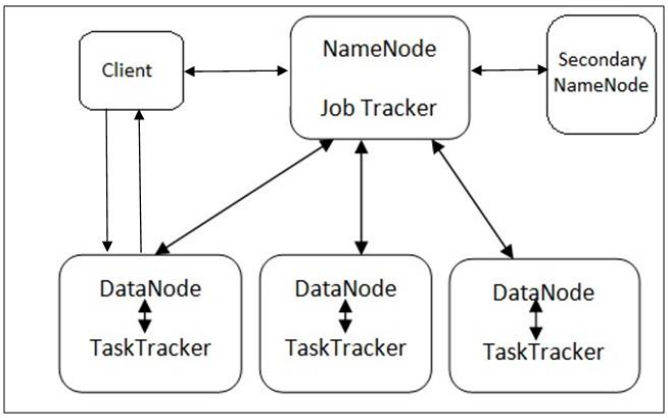

Fig.1. HDFS Architecture

There are five main building blocks of Hadoop [7]. They are (i) NameNode (ii) DataNode (iii) Secondary NameNode (iv) JobTracker (v) TaskTracker. NameNode is the master node on which the JobTracker runs and DataNode is the slave node on which the TaskTracker runs. JobTracker and TaskTracker are the daemons i.e. the process or the services running on background.

3.1 *NameNode*

It is the master of HDFS which directs the slave DataNode to perform the low-level I/O tasks. It keeps a track of how the files are broken down into file blocks and which nodes are used to store those file blocks. The metadata is maintained in the main memory of the NameNode to ensure fast access to the client, on read/write requests. The NameNode is a single point of failure of your Hadoop cluster and this is the only negative aspect to the importance of the NameNode.

3.2 *DataNode*

It is the slave node which is used to perform the work of the distributed filesystem such as reading and writing HDFS blocks to actual files on the local filesystem. When one wants to read or write a HDFS file, the file is broken into blocks and the NameNode on request from the client will tell the client that on which DataNode each block resides in. Default block size is 64MB. Client communicates directly with the DataNodes to process the local files corresponding to the blocks. A DataNode may also communicate with other DataNodes to replicate its data blocks for redundancy using pipeline process.

3.3 *Secondary NameNode (SNN)*

It is an assistant daemon for monitoring the state of the NameNode. It performs periodic checkpoints. One can restart the NameNode using the checkpoint in case of NameNode failure. Like the NameNode, each cluster has one SNN, which resides on its own machine. Secondary NameNode is not a substitute for the NameNode.

3.4 *Job Tracker*

Job tracker is the daemon on which the NameNode runs. Once one submits the code to the cluster, it determines which files to process, assigns nodes to different tasks, and monitors all tasks as they're running. In case the tasks fail, the task is automatically re-launched by the JobTracker, possibly on a different node, up to a predefined limit of retries. There is only one JobTracker daemon per Hadoop cluster which runs on a server as a master node of the cluster.

3.5 *TaskTracker*

TaskTracker is the daemon on which the DataNode runs. It is responsible for executing the individual tasks that the JobTracker assigns. Although there is a single TaskTracker per slave node, each TaskTracker can spawn multiple Java Virtual Machines to handle many map or reduce tasks in parallel. The TaskTracker constantly keeps on communicating with the JobTracker. In case the JobTracker does not receive a heartbeat from a TaskTracker within a specified amount of time, it assumes that the TaskTracker has crashed and hence it resubmits the corresponding tasks to other nodes in the cluster.

4. HDFS FEATURES

The following are the features of Hadoop Distributed File System:

i. Big data: Big Data is basically defined in term of the 'three V's': volume of data (size), velocity of data (speed), and variety of data (type). It is basically a large volume of unstructured live data. Apache Hadoop is an analytics tool that has been labelled 'big data'.
ii. Large: A HDFS instance may consist of thousands of server machines, each storing part of the file system's data
iii. Replication: Each data block is replicated many times (default is 3)
iv. Failure: Failure is the norm rather than exception
v. Fault Tolerance: Detection of faults and quick, automatic recovery from them is a core architectural goal of HDFS.Namenode is consistently checking Datanodes

5. SMALL FILE PROBLEM

Storing large number of small files into HDFS is an overhead in terms of memory usage. Because 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[11]. Further, the size of main memory in NameNode restricts the number of files that can be stored into HDFS.

HDFS does not work well with lots of small files because of the following reasons[8]:
(i) Each block holds a single file, and hence a lot of small blocks (smaller than the configured block size). Reading these entire blocks one by one consumes a lot of time.
(ii) It is the duty of the NameNode to keep a track of each file and each block and store this data into memory. More memory space is occupied by a large number of files.

6. WAYS TO DEAL WITH SMALL FILE PROBLEM

In order to deal with small file problem, following approaches is discussed in this section.

6.1 ***HAR*** [11][14]

HAR stands for Hadoop Archive. It merges small files into large file so as to access the original files in parallel transparently (without expanding the files) and efficiently. HAR files work by building a layered file system on top of HDFS. At the current time HARs are probably best used purely for archival purposes.

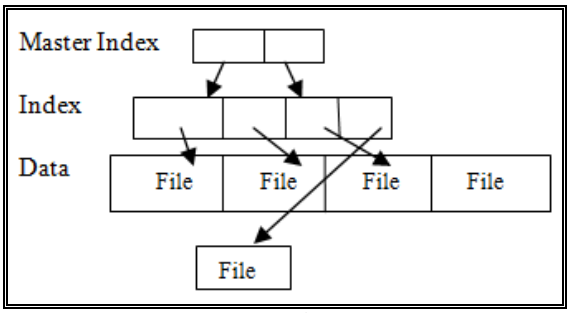

Fig.2 Layout of HAR File

There are certain disadvantages also of HAR files. They are as follows:

i. Reading through files in HAR is slow as compared to reading the files in HDFS. This is because each HAR file access requires two index file reads as well as the data file read.
ii. Inspite of the fact that HAR files are used as the input to MapReduce, no special mechanism allows the maps to operate over all the files in the HAR residing on a HDFS block.

6.2 *Sequence File*[11]

In this technique, the filename is used as a key and its contents as the value. A number of files are put in a single Sequence file and then processing of such files are done using a MapReduce technique. A program can be written for the same. Sequence files can be split and is it considered to be one of the advantage of it. MapReduce can break them into chunks and operate on each chunk independently. The sequence files allows compression.

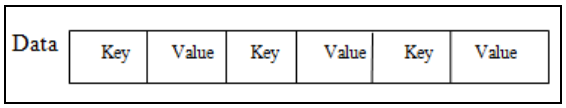

Fig.3 SequenceFile File Layout

Few additional features of SequenceFiles are as follows:

i. Converting existing data to SequenceFile is slow. It is perfectly possible to create a collection of SequenceFiles in parallel. As a possible solution it's best to design data pipeline to write the data at source direct into a SequenceFile, if possible, rather than writing to small files as an intermediate step.
ii. Unlike HAR files there is no way to list all the keys in a SequenceFile, short of reading through the whole file.
iii. This technique is Java-centric.

6.3 *Consolidator*[10]

It takes the records containing files belonging to the same logical file & merges the files together into larger files. It is possible but not practical to merge all the files into a one large file as then it would be a terabytes sized file. A longer time is taken to run such a huge file. Hence Consolidator has a parameter for "desired file size" where user can define the maximum file size of a merged file. In this way Consolidator balances its speed with the desired size of files. "Desired file size" can be set to some multiples of the HDFS block size so that the input splits are larger to optimize for locality.

6.4 *HBase**[10]***

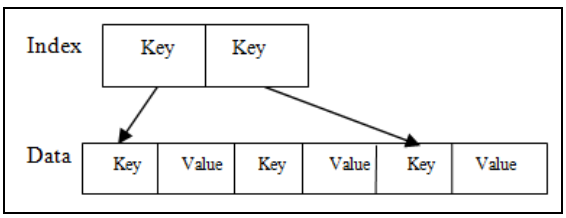

Fig.4 MapFile File Layout

If you have lots of small files, then, depending on the access pattern, a different type of storage might be more appropriate. HBase stores data in MapFiles (indexed SequenceFiles), and is a good choice if you need to do MapReduce style streaming analyses with the occasional random look up. If we are storing lots of small files then HBASE provides a better interface for faster look up of files.

7. CONCLUSION

HDFS is designed to store large files and suffers performance penalty while storing large amount of small files and performing analysis of them. The number of mappers are increased for too many small files and high memory usage is caused by huge numbers of files. Small file problems are solved by given approaches. Each of these approach applicable in different context which improves the efficiency of access to small files in HDFS

References

[1] Hadoop official site, http://hadoop.apache.org/core/.
[2] Shvachko, K.; Hairong Kuang; Radia, S.; Chansler, R., "The Hadoop Distributed File System," Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on , vol., no., pp.1,10, 3-7 May 2010
[3] SlideShare site, http://www.slideshare.net/.
[4] Hadoop archives, http://hadoop.apache.org/common/docs/current/hadoop_archives.html.
[5] Sequence File Wiki, http://wiki.apache.org/hadoop/SequenceFile.
[6] Chuck Lam, Hadoop in Action, Manning Publications,2011
[7] HDFS. http://hadoop.apache.org/hdfs/, 2010
[8] http://pastiaro.wordpress.com/2013/06/05/solving-the-small-files-problem-in-apache-hadoop-appending-and-merging-in-hdfs/
[9] Doug Cutting, Tom White. Hadoop: The Definitive Guide [M]. O'Reilly Media, Yahoo! Press,2011
[10] Small file problem, http://snowplowanalytics.com/blog/2013/05/30/dealing-with-hadoops-small-files-problem/
[11] Tom White, “The Small Files Problem”. http://www.cloudera.com/blog/2009/02/the small files problem, 2009.
[12] Grant Mackey, Saba Sehrish, Jun Wang,"Improving Metadata Management for Small Files in HDFS" IEEExplore,2009
[13] 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
[14] Vaibhav Gopal Korat, Kumar Swamy Pamu, " Reduction of Data at Namenode in HDFS using harballing T echnique" International Journal of Advanced Research in Computer Engineering & Technology,Volume 1, Issue 4, June 2012