ISSN (Online) : 2319 - 8753
ISSN (Print) : 2347 - 6710

International Journal of Innovative Research in Science, Engineering and Technology

Volume 3, Special Issue 3, March 2014

**2014 International Conference on Innovations in Engineering and Technology (ICIET'14)
On 21st& 22nd March Organized by**

K.L.N. College of Engineering, Madurai, Tamil Nadu, India

Hadoop File System with Elastic Replication Management: An Overview

Mamatha S R[#1], Saheli G S[#2], Rajesh R[#3], [*]Arti Arya[#4]

[#] Department of MCA, PESIT Bangalore South Campus, Bangalore, India

ABSTRACT— This paper gives an overview of how Hadoop File System manages massive data as well as handles small files. As data is exponentially pouring in from all sides in all domains, it has become a necessity to manage and analyze such huge amount of data to extract useful information. This huge amount of data is technically termed as Big Data, which in turn falls under Data Science. Currently a lot of research is going on how to handle such vast pool of data. The Apache Hadoop is a software framework that uses simple programming paradigm to process and analyze large data sets(Big Data) across clusters of computers. The Hadoop Distributed File System(HDFS) is one such technology that manages the Big Data efficiently. In this paper, an insight of "how HDFS handles big as well as small amount of data" is presented, reviewed and analyzed. As a result, summarized limitations of existing sys- tems are described in the paper along with the future scope of it.

KEYWORDS—Hadoop, Hadoop Distributed File System, Small files, Elastic Replication Man- agement System, Big Data.

I. INTRODUCTION

Data is getting accumulated from all directions, all domains and in all dimensions, whether its the field of science, technology, public health-care, sports, entertainment, business or any other potential area. The amount of data has become multi-folds in last few years. The term "Big Data" came into existence somewhere in 1999 by Bryson. But during that time 300GB of data was termed as big. There is no specific quantification of big data. Nowadays, generally data running into petabytes (10^5 bytes or 1000 terabytes) and exabytes (10^{18} bytes or 1000 petabytes) is termed as big.This data is knowledge mine, provided knowledge is churned out of it efficiently. For churning this data, one needs efficient data handling techniques. As data is exploding, so normal traditional ways of confronting this data are not efficient. To overcome this problem, some technologies have emerged in last few years to handle this big data. One such technology is Hadoop. Storage, Management and Processing capabilities of Big Data are handled through HDFS, MapReduce[1] and Apache Hadoop as a whole.

Apache Hadoop is an open-source software framework that supports data-intensive distributed applications. Hadoop was derived from Google's MapReduce and Google File System (GFS)[2].The Hadoop framework transparently provides both reliability and data motion to applications. Hadoop implements a computational paradigm named MapReduce, where the application is divided into many small portions of work, each of which may be executed or re-executed on any node in the cluster[11]. In addition, it provides a distributed file system that stores data on the computer nodes, providing very high aggregate bandwidth across the cluster. Both map/reduce and the distributed file

system are designed so that node failures are automatically handled by the framework [2][3]. It enables applications to work with thousands of computation-independent computers and petabytes of data. The en- tire Apache Hadoop "platform" consists of the Hadoop kernel, MapReduce and Hadoop Distributed File Sys- tem(File system designed for storing very large files with streaming data access patterns, running on clus- ters of commodity hardware [4]), as well as a num- ber of related projects including Apache Hive, Apache HBase, and others [5]. The important characteristics of Hadoop are:

- Partitioning of data
- Computations across many hosts
- Executing application computations in parallel close to their data.

The rest of the paper is organized as follows: In Section 2, the general architecture of HDFS is discussed. Section 3 throws light on Elastic Replication Management System. In Section 4, small files handling by HDFS is discussed. Conclusion is presented in section 5.

II. Architecture of Hadoop Distributed File System (HDFS)

An HDFS cluster[11] has two types of node operating in a master-worker pattern: a name- node (the master) and a number of datanodes (workers). The namenode manages the filesystem namespace and stores the filesystem meta-data.[4]

The HDFS namespace is a tree structure of files and directories that are represented on the Namenode with the help of Inodes. Inodes record attributes such as, permissions, modification and access time, namespace and disk space quotas. HDFS stores the entire space in RAM. An HDFS client who wants to read a file contacts the Namenode for location of the data blocks and then reads block content from Datanode that is close to HDFS client. While writing data, the client requests Namenode to select a suite of three Datanodes (typically three, but user selectable file-by-file) to host block replica. The client writes data to Datanode in a pipeline fashion. HDFS is designed in such a way that, it has a single Namenode for each cluster and this cluster can have thousand of Datanodes and ten thousand of HDFS clients per cluster. This single Datanode executes multiple application concurrently.

Datanode in HDFS stores the application data in the form of files. The file content is split into large blocks and each block is independently replicated at multiple Datanodes. This block is represented by two files in the local host's native filesystem. The first file contains the data itself and the second file is the block's metadata. During boot-up Datanode connects to the Namenode to perform a Handshake, which verifies the Namespace ID and the software version of the Datanode. The Datanode automatically shuts down, if either does not match with the Namenode.

Datanodes send Heartbeat message to the Namenode to check the availability of block replicas and confirm the operation of Datanode. In the absence of heart-beat, Namenode considers the Datanode to be out of service and block replicas hosted by that Datanode to be unavailable.

In addition to this, Heartbeat also carry information about total storage capacity, fraction of storage in use and the number of data transfer currently in progress, which are used for Namenode space allocation and load balancing decisions. Namenode does not directly call Datanodes, it uses replies to heartbeats to send instructions to Datanodes.

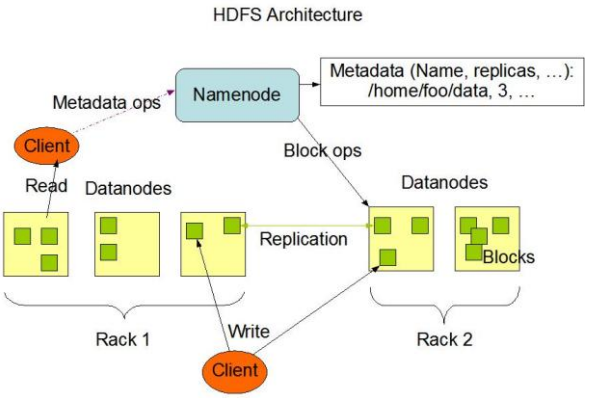

Figure 1: HDFS Architecture[12].

III. Elastic Replication Management System

HDFS is a Distributed file system, that stores large volume of data(Big Data) across clusters of computer reliably and transfers these data sets to variety of applications at a higher bandwidth and provides higher performance and availability by replicating the data blocks(typically 3 replicas). The demand for data popularity changes drastically over time. To get the better performance and higher disk utilization, the HDFS replication policy should be elastic(flexible) and adaptable to data popularity.

ERMS(Elastic Replication Management System) uses complex event processing engine that categorizes the real-time data types(hot/cold data) and depending on this, it increases the replication factor for hot

data(frequency of accessing the data will be more) and removes these extra replicated data blocks, when data cools down(cooled data - frequency of accessing data will be less) and uses erasure codes to deal with cold data.

Depending on the accessing pattern and popularity of data in HDFS, it can be classified into 4 types: hot data, cold data, cooled data and normal data. Hot data is also called as popular data, where the data not only accessed concurrently, but also intensity of data access will be more. Once, when the concurrent access and intensity of data access drops down, such data is termed as Cooled data.

The Data that are occasionally accessed are called as Cold data. Rest all data in HDFS are Normal data.

The popularity of data will be at peak at the time when the data is created newly(fresh). As the time goes by, the popularity nadirs(slows down). The process of data is as follows: Once, the data is created, it will be termed as hot data because the requests for such data will be more from applications. When these requests are processed and completed, it becomes cooled. And then, it becomes normal data. If they are accessed occasionally, the data will become cold. The process of data can be shown in the below diagram, how the accessing and popularity changes as the time goes by.

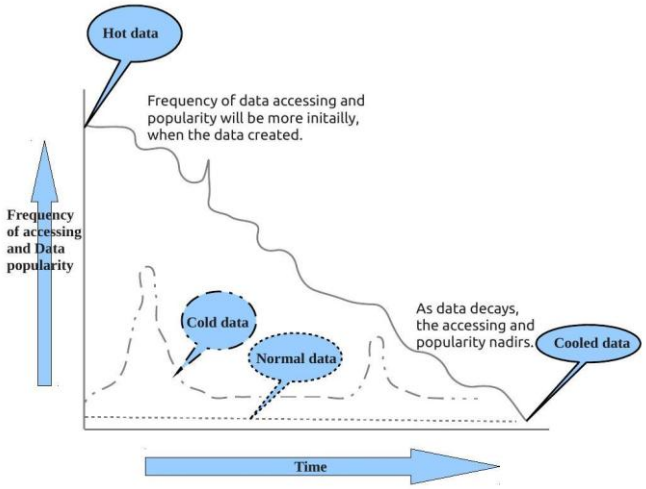

Figure 2: Data Popularity. It shows how the popularity of the data changes as time progresses.

A. Triplication Policy

Generally, HDFS replicates 3 blocks of data(Triplication policy) and will be stored in Data nodes. This triplication policy is well suited in terms of high performance and reliability across clusters of computers. It also ensures that data will be available all the time and fault tolerance during data or disk failures. However, there are 2 drawbacks in this policy:

First, since hot data is requested by many distributed applications simultaneously, this policy does not hold good. In other words, when request numbers are more than three replicas, some of them have to access the data remotely and may compete for the same replica.

Second, this policy does not hold good for cold data also. Since, the intensity of accessing such data is very less, thereby increases the overhead cost on storage management and network bandwidth.

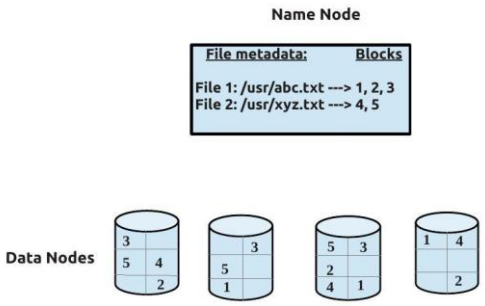

Figure 3: Architecture of Triplication policy in HDFS.

B. About ERMS

ERMS(Elastic Replication Management System) is one way to overcome these problems. It increases the replicas for hot data whenever it is necessary by distributing them reasonably across the data nodes for load balancing and reducing the default replication factor(typically less than three replicas) for cold data by using erasure code technique, so that, management cost will be reduced and provides high availability and performance.

Erasure code is based on the RAID software(redundant array of independent disks), which focuses on providing the high reliable storage with least storage cost.

C. Active and Standby nodes

ERMS introduces an active/standby storage paradigm, that uses the engine called complex event processing(CEP) to differentiate real-time data types, and provides an elastic replication policy for different types

of data[13].It uses a technique called Condor, which is used to increase the replicas for hot data in standby nodes, and removes these extra replicas once when the data cools down. As data becomes cold, erasure codes are used to save the storage management (Removing extra replicas) and network bandwidth. In addition, keeping all the nodes active, might require extra energy to be consumed. However, this model allows the extra nodes to be resided in standby nodes, so that it does not require any extra energy to be used.
Generally, in a HDFS environment, the active nodes will always be busy. To tackle the sudden increase in requests for hot data, ERMS activates the standby nodes and places all the extra replicas at these nodes. When active nodes are busy, the standby nodes play a vital role by initiating itself to service the requests in place of active nodes.

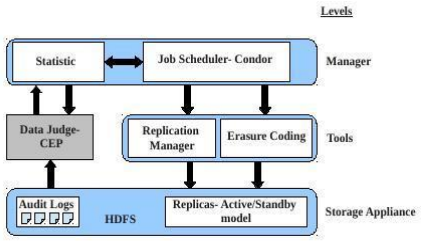

Figure 4: Active and Standby Nodes

D. ERMS Architecture

ERMS Architecture is made up of number of components, which are arranged in three different levels.
The Data Judge module, uses the log file from the HDFS clusters and uses CEP to differentiate the real-time data types(hot/cold data). After differentiating the real-time data, the manager level of ERMS uses the Condor to schedule the replication manager tool and erasure coding tool to manage the replicas of data. HDFS is the basic storage appliance where all the audit log files and replicas are stored.
Thus from this technique, the Replication factor can be dynamic in real time, which reduces the overhead cost of storage management in terms of replicas and as well as the provides high availability and performance.

IV. SMALL FILES IN HDFS

HDFS is expert in handling large files of data, but dealing with small files is inefficient, a large number of small files occupies large amount of system memory and even accessing small files is hundred times more than accessing large files. So repository and managing massive small files in HDFS is a practical problem. The reasons for handling small files are:

1. Even though HDFS is basically designed for accumulating large files, it also retrieves small files of lower efficiency. Small files send request to Namenode several times in which Namenode has a greater overhead and even file addressing cost is high. Due to this, original intention of HDFS will be infringed.

2. Memory usage of data node and Namenode will be very high as the DataNode will store file block in memory and the small file occupy a file chunk.

To solve the small file problem, method of file merger is used.

1. HAR (Hadoop Archive)[9] tool is used, but this method occupied large amount of disk resources.

2. Another method is to add a small file processing module in the original HDFS system which when small file arrives will be directly delivered to processing module.

Due to the drawbacks of these two methods a novel approach to handle small files is introduced, which will not change the present HDFS system but improves its efficiency while handling small files. Retrieval and fetching of small files is done by an engine which builds the file index and file process procedure.

Design Scheme for Engine of Small File:

The small file engine processor uses Reactor Multiprocessor I/O, which process many request task. This engine has 2 components:

a) Small files written and merger

b) Small file read and separation[10]

a) Small files written and merger - This component establishes a TCP server using local port to complete the file merger tasks. Then service waits for the small file merger request that uses the observer pattern and the events gets registered on listening socket.

Once the request arrives, the listening socket state gets changed and the service adds the file index to the head of small file immediately which later

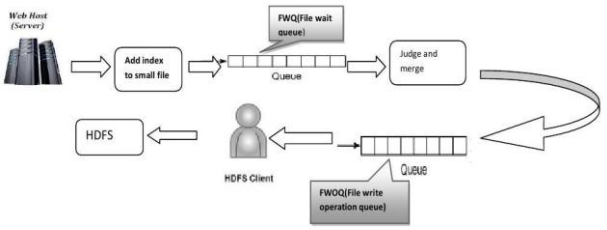

Figure 5: Writing Small files and Merger.

forms a new small file, and will be delivered to a file wait queue. A new file wait queue(FWQ) is created, which adds the new file to this queue. Threads are allocated to read files from FWQ, which is thread-safe. Here multiple small files , merge to form a new file chunk and each merger calculates the current file block size. Once the files are merged it is added to the FWOQ(File Write Operation Queue)and from FWOQ files will be taken to HDFS and files are written here. Even the writing process uses multiple threads.

b) Small file read and separation - This component establishes a TCP server which accomplishes file separation task .Then service waits for the small file write request.Once the request arrives ,the multiple threads get merge file from HDFS and adds to queue FRWQ(File Read Write Queue).The file block will be read from FRWQ and will be split into many individual small files. These files will be flung into FCT(File Cache Table) which is implemented using Hash Table. Here small file index is the key,and the file content is the value. The files get retrieved from FCT. Abstaining files which are not retrieved from FCT for long time can be done by using LRU replacement algorithm. Hash table (FAT) will be procedure created to record the access times of files.

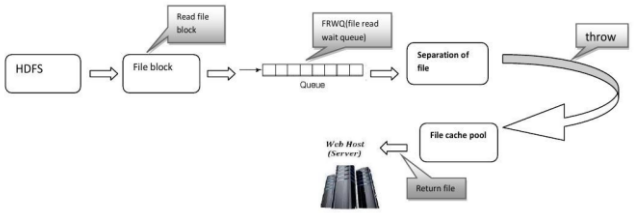

Figure 6: Reading Small files and Separation.

The engine, which is applied on HDFS system, can even be used to other distributed file system.

A. Novel Indexing Scheme for handling small files in HDFS

Large file storage and maintenance is efficiently done by HDFS. NameNode is responsible to manage all the files stored in HDFS. This is done by storing metadata of all files in the main memory of NameNode. Storing large number of small files is a constraint on HDFS main memory. HDFS does not correlate the files and hence does not provide any prefetching mechanism [6]. The paper surveyed further proposes a solution to improve the storage and management efficiencies of small files in HDFS, based on the concept of Extended Hadoop Distributed File System (EHDFS by Dong et al.)[7]. The concept behind EHDFS is to bind interrelated small files into a large file, which reduces the metadata footprint on NameNode. The small files can be accessed by using an indexing mechanism. The efficiency of handling small file requests is achieved by providing file and index prefetching mechanisms based on interrelationships between files. This will reduce the burden on Name Nodes main memory that is created due to large number of metadata requests.

Small Files Problem:

When a single Name Node stores metadata of all the files in its main memory, it becomes a bottleneck to manage such a huge amount of small files. A small file is usually defined in terms of size, if it is relatively lesser than the HDFS block size[6]. This issue is known as the small files problem[8]. The small files problem can be better understood with the following explanation.

The complete metadata set is stored in main memory of Name Node for faster access. Each data block has its own metadata. If the file size is greater than the block size, then the metadata volume stored is substantiated. When in fact if the file size is smaller than the block size, each file occupies a block and for large number of small files, the extent of metadata stored is remarkably huge. This can be better understood with an illustration. Let us assume that, the metadata for each file block in main memory takes about 150 bytes. Consequently, for a 1GB file, divided into 16 64MB blocks, 2.4KB of metadata is stored. While, for 10500 files of size 100KB each (total of 1GB), about 1.5MB metadata is stored[6][8]. Thus, though a good amount of small files occupy minimum space in file system, its corresponding metadata consumes a great magnitude of main memory in Name Node. Additionally, accessing and managing such large amount of files creates

congestion in Name Node, thus results in poor performance of HDFS.

The design is based on the concept of Extended Hadoop Distributed File System (EHDFS)[7], which renders two significant services that are as follows, an enhanced indexing mechanism and prefetching of indexing information. In the EHDFS approach, the small files problem can be potentially vanquished using 4 techniques.
They are :

- File merging,
- File mapping,
- Prefetching and
- File extraction.

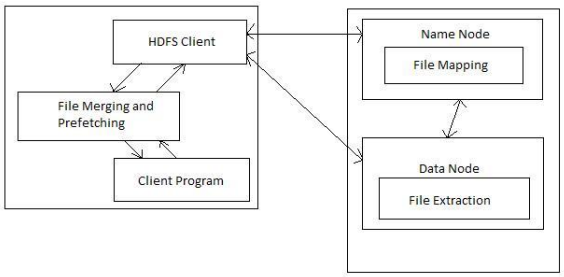

Figure 7: Small file handling operations

When a large number of small files are to be managed by HDFS, the performance of the system would be at stake and also it imposes a large amount of metadata footprint on the Name Node. The paper discussed and reviewed so far proposes a solution that can handle the small files problem efficiently using the concept of EHDFS. In this technique, a large number of small files can be merged into a single file and it also provides a mechanism of prefetching certain file metadata in order to improve the efficiency. The four mechanisms namely, File merging, File mapping, Prefetching and File extraction are carried out in a systematic manner to facilitate the File read and Write operations. Hence it can be concluded that the use of EHDFS principle improves the efficiency of accessing small files and also reduces the metadata footprint on the Name Nodes main memory.

V. CONCLUSION

In this paper, the Hadoop file system is discussed explaining how it handles big files as well as small files efficiently. ERMS plays a major role in handling hot data by distributing the data across the nodes. Using ERMS replication factor becomes dynamic in real time that helps in reducing the overall cost of storing the data in terms of replicas. Also, Extended HDFS can handle the small files that are available in large numbers efficiently. This technique merges the small files into one large file that can be handled by EHDFS easily. Also, four mechanisms, namely File merging, File mapping, Prefetching and file extraction are carried out in a systematic manner to facilitate file read and write operations. So, overall this paper discusses in detail about how Hadoop File System manages both big files and small files efficiently.

REFERENCES

[1] map/reduce. http://wiki.apache.org/hadoop/ MapReduce.
[2] Apache Hadoop. http://en.wikipedia.org/ wiki/Apache_Hadoop.
[3] Hadoop Overview. http://hadoop.apache.org
[4] Hadoop: The Definitive Guide,by Tom White, Published by O'Reilly Media 2012 edition
[5] Hadoop-related projects at hadoop.apache.org.
[6] Chandrasekar S, Dakshinamurthy R, Seshakumar P G, Prabavathy B, Chitra Babu. "*A Novel index- ing scheme for efficient handling of small files in hadoop distributed file system*".In proc. of IEEE Internatioal Conf. on Computer Communication and Informatics, Jan 04-06, 2013, Coimbatore, In- dia.
[7] B.Dong, J.Qin, Q.Zhong, J. Li,Y.Li. "*A Novel approach to improving the efficiency of storing and accessing small files on hadoop*".In Proc. of IEEE International Conference on Services Computing, Miami, Florida, USA, July 2010.
[8] Tom White, "*The small files problem*" http: //www.cloudera.com/blog/2009/02/ the small files problem, 2009.
[9] Hadoop archives, http://hadoop.apache.org/ common/docs/current/hadoop_archives.html.
[10] Yang Zhang,Dan Liu. "Improving the Efficiency of Storing for Small Files in HDFS". In Proc. of 2012 International Conference on Computer Sci- ence and Service System, Nanjing, China.
[11] http://search-hadoop.com/jd/hcommon/org/ apache/hadoop/mapreduce/Cluster.html
[12] http://hadoop.apache.org/docs/stable/ hdfs_design.html
[13] Zhendong Cheng,Zhongzhi Luan, You Meng, Yi- jing Xu, Depei Qian Sino, Alain Roy, Ning Zhang. "ERMS : An Elastic Replication Management System for HDFS". In Proc. of IEEE Interna- tional Conference on Cluster Computing Work- shops, sept 24-28 2012, Beijing, China.