International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 4 Issue 05, May-2015

A Review on Distributed File System in Hadoop

Mr. Amol M. Kadam
PG Student ME Computer Science & Engineering
SIETC, Paniv
Akluj, India

Dr. Pradip K. Deshmukh
Principal Computer Science & Engineering
SIETC, Paniv
Akluj, India

Prof. Prakash B. Dhainje
Vice-Principal, Computer Science & Engineering
SIETC, Paniv
Akluj, India

***Abstract*— When a dataset exceeds the storage capacity of a single physical machine, it becomes require to divide it across a number of separate machines. File systems that manage the storage over a network of machines are called distributed file system. Hadoop meets with a distributed file system called Hadoop Distributed File System (HDFS). HDFS is a file system designed for storing huge files with streaming data access patterns, running on clusters of commodity hardware. HDFS files are hundreds of gigabytes or in terabytes in size. There are Hadoop clusters running currently that store petabytes of data. HDFS is built around the most efficient data processing patterns is a write-once, read-many time patterns.**

Keywords*— *DataNode, Hadoop, HDFS, NameNode

I. INTRODUCTION

Hadoop was originally built by a Yahoo! An Engineer named Doug Cutting and is now an open source project managed by the Apache software Foundation [13]. Hadoop is created to parallelize data processing over computing nodes to speed computations and hide latency.
Hadoop has two primary components [12]:
A. Hadoop Distributed File System- A reliable, low cost, data storage cluster that facilitates the management of equivalent files across machines.
B. MapReduce Engine- A high performance Distributed data processing implementation of the MapReduce algorithm.
Hadoop is designed to process huge amounts of structured and unstructured data (terabytes to petabytes). It is executed on the racks of commodity servers as a Hadoop cluster. Servers can be attached or detached from the cluster dynamically, because Hadoop is designed to be "self-healing". Also, Hadoop is able to identify changes, including failures and adjust to those changes and continue to operate without interruption.

The HDFS is an adaptable, flexible, clustered approach to managing files in a big data environment. HDFS is not the terminal destination for the files. Rather, it is a data service that offers a unique set of capabilities needed when data volumes and velocity are big.

HDFS is an excellent choice for supporting big data analysis. The service includes a "NameNode" and multiple "DataNodes" running on a commodity hardware cluster. It provides the highest levels of performance, when the whole cluster is in the alike physical rack in the data center. In Hadoop cluster, data is distributed over the machines of the cluster when it is loaded.

II. ARCHITECTURE

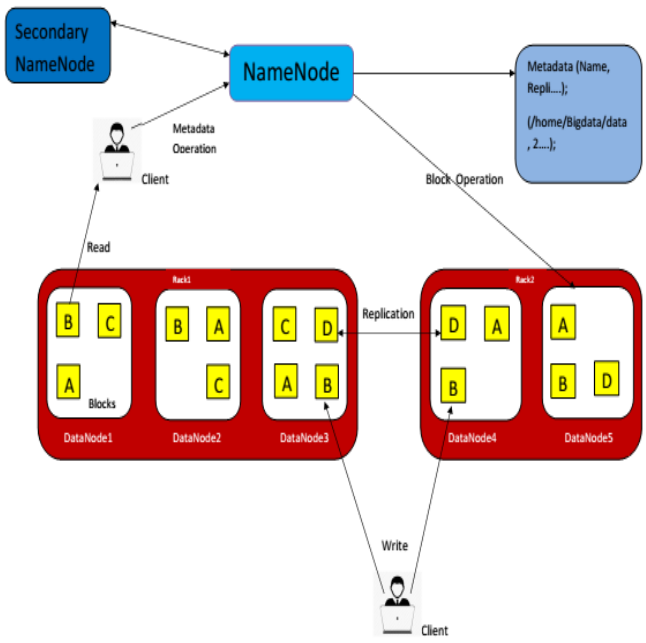

Fig. 1. Architecture of HDFS

A. NameNode-
HDFS works by dividing large files into smaller pieces called blocks. The blocks are stored in the data nodes. It is the responsibility of the NameNode to know what blocks on which data nodes make up the complete file. The NameNode managing all access to the files, including reads, writes, create, deletes, and replication of data blocks on the data nodes. System namespace is the complete collection of all the files in the cluster. NameNode will control this namespace. NameNode and the strong relationship between Datanode's and they operate in a "loosely coupled" mode. In a typical

configuration, you find one NameNode and possibly a data node running on one physical server in the rack. The NameNode is smarter than the Data nodes. NameNode is so critical for correct operation of the cluster; it can and should be replicated to guard against a single point failure.
HDFS break files into a related collection of little blocks. These blocks are distributed among the data nodes in the HDFS cluster and are managed by the NameNode. NameNode uses a "rack ID" to keep track of the data nodes in the cluster.

B. DataNode-
Data nodes are flexible but not smart. Within the HDFS cluster, data blocks are replicated over multiple data nodes and access is managed by the NameNode. Data nodes also provide "heartbeat" messages to detect and ensure connectivity between the NameNode and the data nodes. During normal operation, data-nodes periodically send heartbeats to the name-node to indicate that the data-node is alive. The default heartbeat interval is three seconds. If the name-node does not accept a heartbeat from a data-node in 10 minutes, it considered as the data-node dead and schedules its blocks for replication on other nodes.
To ensure integrity over the cluster, HDFS uses transaction logs and Checksum validation.
The failure of one server may not necessarily corrupt a file, because data blocks are replicated over several data Nodes. All parameters can be adjusted during the operation of the cluster. So, when the cluster is performed in their degree of replication, the number of data Nodes, and HDFS namespaces are accepted.

C. Data replication-
HDFS keeps one replica of every block locally. It places a second replica on a different rack to guard against an entire rack failure. It sends a third replica to the same remote rack, but to a different server in the rack. It can then send additional replicas to random locations in local or remote clusters. Client applications do not need to worry about where all the blocks are situated. To ensure highest performance clients are directed to the nearest replica [10].
To provide high availability of data in high demand is the most important advantage of the replication technique. The reliability and availability points of view replication are important.

D. Secondary NameNode-
NameNode snatch a copy of the NameNode's in memory metadata and files used to store metadata, because secondary NameNode sometimes attached to the NameNode.

E. HDFS Client-
HDFS client can access the file system through user application. HDFS can manage read, write, copy and delete operations.

III. HDFS READ AND WRITE OPERATIONS

A. File Read Operation-
File read operation in HDFS contains six steps [13], as shown in fig. Suppose an HDFS client wants to read a file. So, reading the file contains following steps:

Step1- By calling open () method on a file system object, the client will unlock the file. In which HDFS is an object of Distributed file system class.

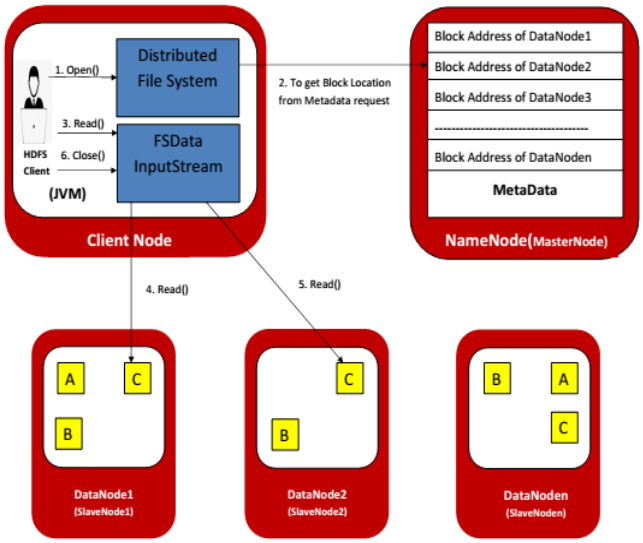

Fig. 2. File read operation in HDFS

Step2- To determine the location of the blocks for the file, Distributed File System calls the NameNode by using RPC. The NameNode will return the addresses of all the DataNodes for each block that has a copy of that block.

Step3- To connect to the first closest DataNode for the first block in the file, the client calls the read () method on the stream (FsDataInputStream).

Step4- Client will call the read () method repeatedly over the stream and DataNode will send streamed data to the client.

Step5- The connection between DataNode and DFSInputStream will be closed when the end of the block is reached. After that it finds the next block on the best DataNode.

Step6- Client read blocks in order through the stream, when the DFSInputStream open the new connection to the DataNodes. Finally, the client will call close () method on the FsDataInputStream, when the client finishes there reading.

B. File Write Operation-
There are seven steps [13] in writing a data to HDFS as shown in fig. 3:

Step1- With the help of Create () method client can create the file on Distributed File System.

Step2- By making an RPC call to the NameNode, Distributed File System creates a new file in the namespace and there is no block associated with it. NameNode will examine that client has the right permission to create the file. NameNode also checks file doesn't already exist. The NameNode makes a record of the new file if these checks pass, else file creation fails. The Client will throw an IOException if the file creation fails. For handling communication between the DataNodes and NameNodes, FSDataOutputStream wraps a DFSOutputStream.

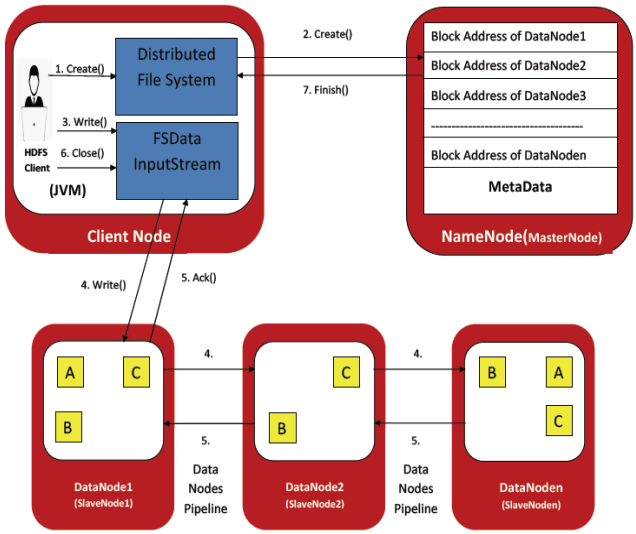

Fig. 3. File writes operation in HDFS

Step3- When the client writes data, DFSOutputStream divide data into different packets. After that it writes to a within queue. The pipeline is created with the list of DataNodes; here we will assume the degree of replication is there, i.e. three nodes in the pipeline.First DataNode streams the packets from DataStreamer in the pipeline. So, stores the packet in the current node and forwards it to the next DataNode in the pipeline.

Step4- similarly packet is stored in second DataNode and forwards it to the next or last DataNode in the pipeline.

Step5- When the DataNodes in the pipeline will send packet acknowledgement. The packet is removed from the acknowledgement queue.

Step6- client calls close () method on the stream, after the client has finished writing data.

Step7- finish () method cleans all the remaining packets to the DataNode pipeline.

IV. HDFS FILE PERMISSIONS

For files and directories, HDFS has supplied different permission models.
There are three types of permission [12]: the read permission (r), the write permission (w), and the execute permission (x).

To read files or list the contents of a directory the read permission is necessary. To write a file, to create or delete files in it the write permission is necessary. The execute permission is avoided for a file since you can't execute a file on HDFS, and for a directory it is necessary to access its children.
Each file and directory has an owner, a group user, and other user. The mode is build up of the permissions for the user who is the owner, the permissions for the users who are members of the group, and the permissions for users who are not the owners and members of the group.

V. HADOOP AND NETWORK TOPOLOGY

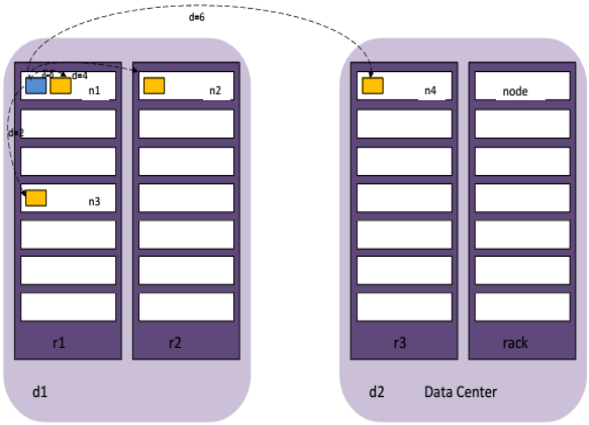

Fig. 4. Network distance in Hadoop

Hadoop network is represented as a tree and the distance between two nodes is the sum of their distances to their closest common ancestor [13]. Levels in the tree are not affecting, but it is common to have levels that correlate to the node, the rack, and the data center that a process is running on.
For each of the following scenarios, less bandwidth is available:

- Processes on the identical node
- Distinct nodes in the identical rack
- Nodes in distinct racks in the identical data Center
- Nodes in distinct data centers.

This is illustrated diagrammatically in fig. 4
For example [12], imagine a node n1 on rack r1 in data center d1. This can be expressed as /d1/r1/n1. Using this notations, here are the distances for the four scenarios:

- Distance (/d1/r1/n1, /d1/r1/n1) = 0 (Processes on the identified node)
- Distance (/d1/r1/n1, /d1/r1/n2) = 2 (distinct Nodes in the identical rack)
- Distance (/d1/r1/n1, /d1/r2/n3) = 4 (nodes On distinct racks in the identical data center)
- Distance (/d1/r1/n1, /d2/r3/n4) = 6 (nodes in Distinct data centers)

VI. ALGORITHM

1: Mapper Class
2: method Map(docuid a; docu d)
3: for all term trm ε docu d do
4: Emit(term trm; count 1)

1: Reducer Class
2: method Reduce(term trm; counts [cnt1; cnt2; : :])
3: sum=0
4: for all count cnt ε counts [cnt1; cnt2; : :] do
5: sum=sum + cnt
6: Emit(term trm; count sum)

VII. RESULTS

A. *Fig.5 shows Adding DataNode in Hadoop*

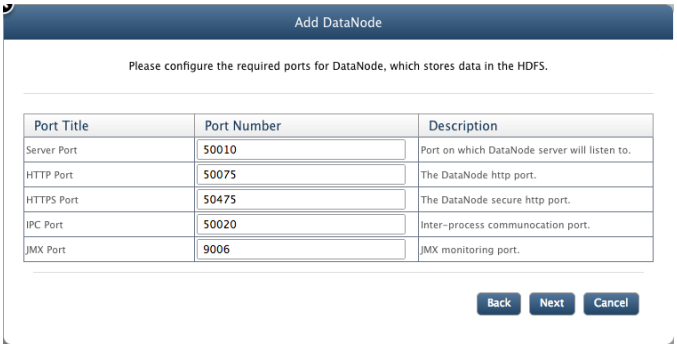

Fig. 5. Hadoop DataNode

B. *Fig.6. shows Hadoop NameNode and Cluster information*

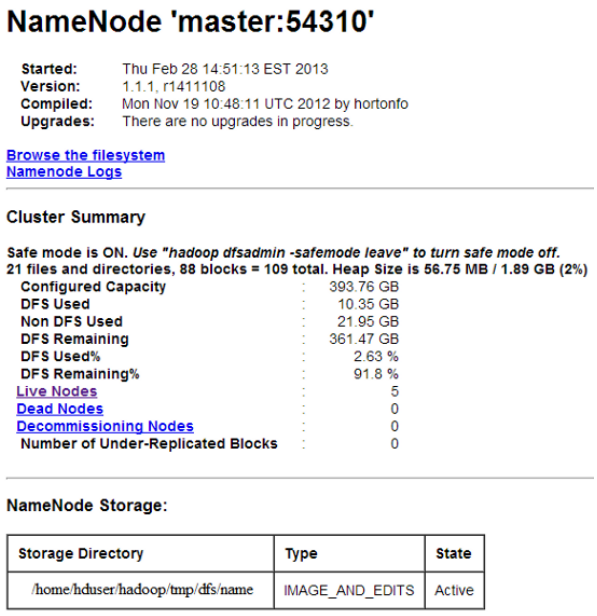

Fig. 6. Hadoop NameNode and cluster

C. *Fig.7 shows Installing single node in Hadoop*

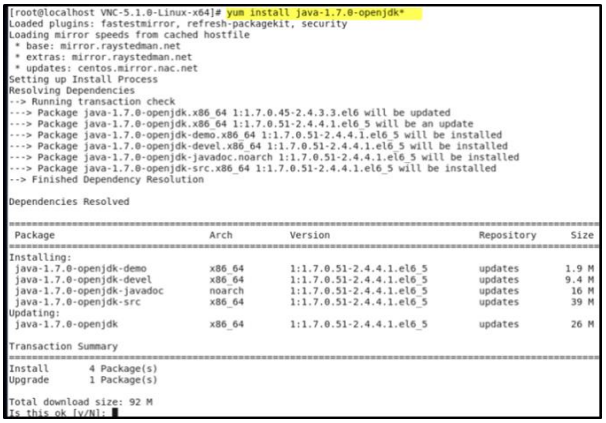

Fig. 7. Installation of Node

D. *Fig. 8 shows Data replication factor in Hadoop*

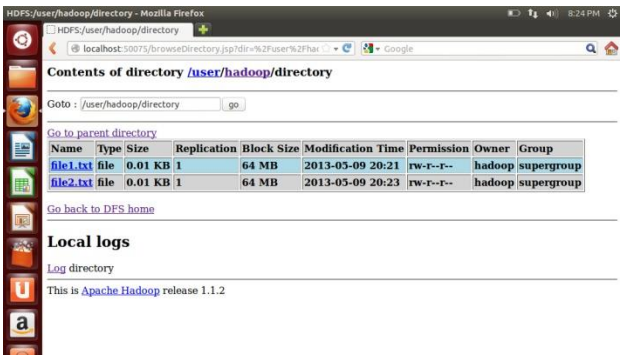

Fig. 8. Data replication factor

E. *Fig. 9 shows file permissions for Hadoop file*

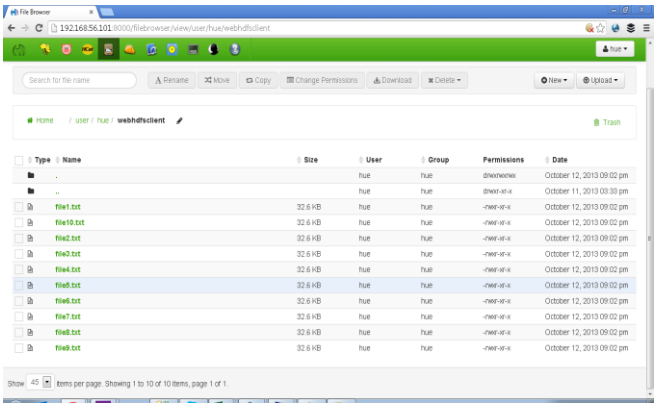

Fig. 9. File permissions for Hadoop

F. *Fig. 10 shows Selecting network and topology in Hadoop*

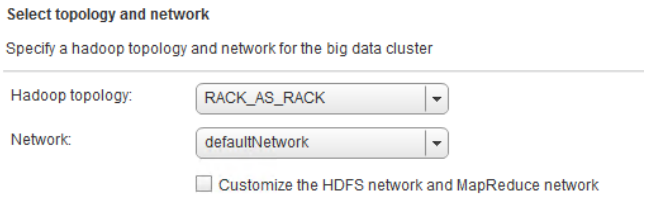

Fig. 10. Selecting topology and network

VIII. CONCLUSION

A high throughput access to the data of an application is provided by using Hadoop distributed file system. HDFS designed to carry petabytes of data with high fault tolerance. petabytes of data are saved redundantly over several numbers of servers or machines. MapReduce model is used to process data stored in HDFS. We can analyze and count the number of words in a large data file by using HDFS.

REFERENCES

[1] Apache Hadoop. http://hadoop.apache.org/
[2] www.bigdataplanet.info
[3] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler,"Hadoop Distributed File System",2010
[4] Haojun Liao, Jizhong Han, Jinyun Fang "Multi-Dimensional Index on Hadoop Distributed File System", Fifth IEEE International Conference on Networking, Architecture, and Storage, 2010
[5] Kyriacos Talattinis, Aikaterini Sidiropoulou, Konstantinos Chalkias, and George Stephanides, "Parallel Collection of Live Data Using Hadoop", in 14th Panhellenic Conference on Informatics, 2010
[6] Shafer, J, "The Hadoop Distributed File System: Balancing Portability and Performance", 2010
[7] Zhi-Dan Zhao, " User-Based collaborative Filtering Fig 10: users on HDFS Recommendation Algorithms on Hadoop", 2010
[8] Rini T. Kaushik, Milind Bhandarkar, "Evolution and Analysis of GreenHDFS",2010.
[9] Garhan Attebury, Andrew Baranovski, "Hadoop Distributed File. The Fig 11 shows the Data Chunks distributed over the System for Grid",2009
[10] K. V. Shvachko, "HDFS Scalability: The limits to growth,"; login:.April 2010,pp. 6–9.
[11] Jeffrey Dean and Sanjay Ghemawat "MapReduce: Simplified Data Processing on Large Clusters "GoogleInc.
[12] Tom White, "Hadoop The Definitive Guide", 2nd ed., O'REILLY, 2011, pp. 41–73.
[13] Pooja S.Honnutagi, "The Hadoop distributed file system", International Journal of Computer Science and Information Technology, Vol.5(5), 2014, pp. 6238-6243.