© 2017 IJSRST | Volume 3 | Issue 6 | Print ISSN: 2395-6011 | Online ISSN: 2395-602X
Themed Section: Science and Technology

The role of Hadoop Distributed Files System(HDFS) in Technological era

Navjot Jyoti
Assistant Professor, Northwest Group of Institutions, Dhudike (Moga), Punjab, India

ABSTRACT

The popularity of cloud computing has been growing steadily for several years. It has been used more and more widely in many works of life. Cloud computing provides massive clusters for efficient large scale computation and data analysis. The MapReduce structure and its open source execution Hadoop have set up themselves as one of the most poplular large informational collections analyzers. They are generally utilized by numerous cloud specialist organizations, for example, Amazon EC2 Cloud.

Keywords : MapReduce, HDFS, GFS

I. INTRODUCTION

Hadoop is an open source framework, facilitated by the Apache Software Foundation that gives a solid, fault tolerant, distributed document system and application programming interfaces. These empower its map-reduce framework to analyze substantial volumes of information in parallel. Straightforwardness of the Hadoop programming model takes into consideration clear usage of numerous applications. Java applications have the most direct access, yet Hadoop likewise has spilling capacities that take into account usage in any language.

Several organizations that need to handle large amounts of data are using map-reduce implementations to manage that data. Google started using a map-reduce system internally before 2004 [Dean 2004][1]. Yahoo runs the largest Hadoop cluster, running over a Linux cluster of over 10,000 cores [Yahoo 2008]. Vendors, such as Amazon, utilize Hadoop as part of their cloud computing service.

II. Hadoop Distributed File System

The Hadoop Distributed Files System (HDFS) keeps running over a local document framework and is just open through the Hadoop Application Programming Interfaces (APIs). HDFS arrangements appropriate information in similarly estimated pieces over the accessible information hubs. This division of information works best for huge records that can be put away as products of the lumping size designed for the HDFS.

In the event that the documents are littler than the lumping size, the HDFS will squander nearby record framework assets with exhaust, dispensed bytes. Excess and adaptation to internal failure are accomplished by recreating these pieces on different hubs. Hadoop endeavors to run the guide activities on duplicates of the information neighborhood the mapping undertaking. This decreases the measure of information that should be moved around. A HDFS may comprise of hundreds or thousands of server machines, each consisted of storing piece of the file information.

Along these lines, recognition of flaws and speedy, programmed recuperation from them is a center design objective of HDFS. Applications that keep running on HDFS require streaming access to their data collections. They are not universally useful applications that regularly keep running on broadly useful document frameworks. HDFS is composed more for cluster preparing as opposed to intelligent use by clients [2].

III. Map-Reduce API

Hadoop uncovered three activities for executing the map reduce algorithm, mapping, joining, and decreasing. The framework is executed in Java; in any case, Hadoop additionally uncovered a gushing interface that permits programs written in any language to process every activity.

The need for real-time execution of MapReduce applications is increasingly arising and the effective processing strategy for deadline jobs has drawn attention of lots of researchers[3].

The process of MapReduce includes two major parts, Map function and Reduce function. The input files will be automatically split and copied to different computing nodes. After that the inputs will be sent to Map function in key-value pair format. The Map function will process the input pairs and also generate intermediate key-value pairs as inputs for Reduce function. The Reduce function will combine the inputs who have the same key and generate the final result. The final result will be written into the distributed file system[4]. The users only need to implement Map and Reduce functions. They do not need to concern about how to partition the input which is automatically done by the model. The model will make tasks assigned evenly to every machine. Sometimes, to reduce communication overhead, a optional part Combine function can be employed. The Combine function can be regarded as a local Reduce, combines the results from Map function together locally, and returns a single value for Reduce function.

IV. HDFS Architecture

HDFS has a master/slave architecture. HDFS has an ace/slave engineering.[5]A HDFS cluster comprises of a solitary NameNode, a master server that deals with the file system namespace and manages access to files by users. Also there are various DataNodes, typically one for every cluster in the group, which manage capacity appended to the hubs that they keep running on. HDFS uncovered a document framework namespace and enables client information to be put away in files. Inside, a file is part into at least one squares and these pieces are put away in an arrangement of DataNodes. The NameNode executes file framework namespace operations like opening, closing, and renaming files and directories. It additionally decides the mapping of pieces to DataNodes. The DataNodes are in charge of serving read and write requests from the file system's clients.The DataNodes additionally perform square creation, deletion, and replication upon direction from the NameNode.

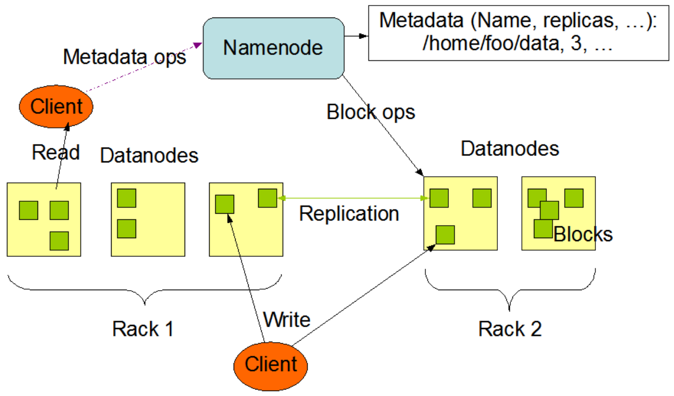

Fig 1 : HDFS Architecture

The NameNode and DataNode are bits of programming intended to keep running on item machines. These machines commonly run a GNU/Linux working framework (OS). HDFS is assembled utilizing the Java language; any machine that backings Java can run the NameNode or the DataNode programming. Use of the exceptionally versatile Java language implies that HDFS can be sent on an extensive variety of machines. A normal sending has a committed machine that runs just the NameNode programming. Each of alternate machines in the bunch runs one case of the DataNode programming. The architecture does not block running different DataNodes on a similar machine however in a genuine arrangement that is once in a while the case.

The presence of a single NameNode in a group incredibly streamlines the design of the framework. The NameNode is the judge and repository for all HDFS metadata. The framework is planned such that client information never flows through the NameNode.

Functions of NameNode:

- It is the master daemon that maintains and manages the DataNodes (slave nodes)
- It records the metadata of all the files stored in the cluster, e.g. The location of blocks stored,the size of the files, permissions, hierarchy, etc. There are two files associated with the metadata:
  - **FsImage:** It contains the complete state of the file system namespace since the start of the NameNode.
  - **EditLogs:** It contains all the recent modifications made to the file system with respect to the most recent FsImage.
- It records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog.
- It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live.
- It keeps a record of all the blocks in HDFS and in which nodes these blocks are located.
- The NameNode is also responsible to take care of the **replication factor** of all the blocks which we will discuss in detail later in this HDFS tutorial blog.
- In **case of the DataNode failure**, the NameNode chooses new DataNodes for new replicas,balance disk usage and manages the communication traffic to the DataNodes.

Functions of DataNode:

- These are slave daemons or process which runs on each slave machine.
- The actual data is stored on DataNodes.
- The DataNodes perform the low-level read and write requests from the file system's clients.
- They send heartbeats to the NameNode periodically to report the overall health of

HDFS, by default, this frequency is set to 3 seconds.

The File System Namespace

HDFS supports a customary hierarchical file association. A user or an application can create directories and store files inside these directories. The file framework namespace chain of command is like most other existing document frameworks; one can make and evacuate documents, move a document starting with one directory then onto the next, or rename a file. HDFS does not yet actualize client standards. HDFS does not support hard connections or delicate connections. In any case, the HDFS architecture does not block executing these highlights.

The NameNode keeps up the record framework namespace[6]. Any change to the file framework namespace or its properties is recorded by the NameNode. An application can indicate the quantity of reproductions of a document that ought to be kept up by HDFS. The quantity of duplicates of a record is known as the replication factor of that document. This data is put away by the NameNode.

Data Replication

HDFS is intended to dependably store substantial records crosswise over machines in an expansive bunch. It stores each record as a grouping of blocks; all blocks in a file aside from the last block are a similar size. The pieces of a file are replicated for adaptation to internal failure. The block size and replication factor are configurable per document. An application can determine the quantity of copies of a document. The replication factor can be indicated at files creation time and can be changed later. Files in HDFS are write once and have entirely one writer at a time.

The NameNode settles on all choices with respect to replication of pieces. It intermittently gets a Heartbeat and a Blockreport from each of the DataNodes in the bunch. Receipt of a Heartbeat infers that the DataNode is working properly. A Blockreport contains a list of all blocks on a DataNode.

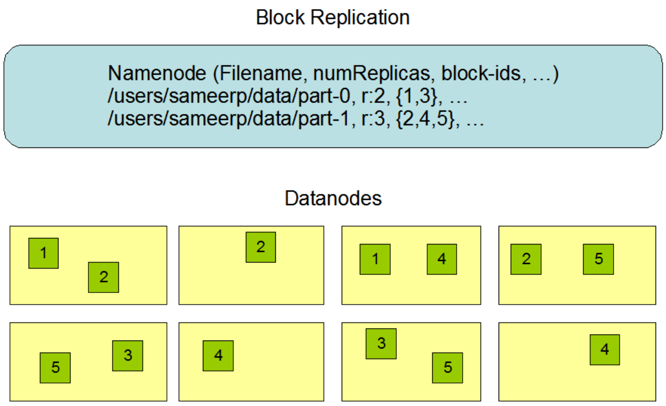

Fig 2 : Data replication

Balancer

HDFS block placement strategy does not consider DataNode plate space usage. This is to abstain from putting new information at a little subset of the DataNodes with a great deal of free storage. Along these lines data may not generally be put consistently crosswise over DataNodes. Imbalance likewise happens when new hubs are added to the clusters.

The balancer is a tool that adjusts hard disk space usage on a HDFS cluster[7]. The threshold value is taken between 0 and 1. A cluster is changed if, for each DataNode, the use of the node varies from the use of the entire cluster by near to the limit value.

The tool is conveyed as an application program that can be controlled by the cluster administration. It iteratively moves reproductions from DataNodes with higher usage to DataNodes with bring down use. One key necessity for the balancer is to keep up information accessibility. While picking a reproduction to move and choosing its goal, the balancer ensures that the choice does not decrease either the quantity of replicas or the number of racks.

The balancer optimizes the balancing process by minimizing the inter-rack data copying. If the balancer decides that a replica A needs to be moved to a different rack and the destination rack happens to have a replica B of the same block, the data will be copied from replica B instead of replica A. A configuration parameter limits the bandwidth consumed by rebalancing operations. The higher the allowed bandwidth, the faster a cluster can reach the balanced state, but with greater competition with application processes.

The Communication Protocols

All HDFS communication protocols are layered on top of the TCP/IP protocol. A client establishes a connection to a configurable TCP port on the Namenode machine. It talks the ClientProtocol with the Namenode. The Datanodes talk to the Namenode using the DatanodeProtocol. A Remote Procedure Call (RPC) abstraction wraps both the ClientProtocol and the DatanodeProtocol. By design, the Namenode never initiates any RPCs. Instead, it only responds to RPC requests issued by Datanodes or clients.

Robustness

The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are Namenode failures, Datanode failures and network partitions.

Data Disk Failure, Heartbeats and Re-Replication

Each Datanode sends a Heartbeat message to the Namenode periodically. A network partition can cause a subset of Datanodes to lose connectivity with the Namenode. The Namenode detects this condition by the absence of a Heartbeat message. The Namenode marks Datanodes without recent Heartbeats as dead and does not forward any new IO requests to them. Any data that was registered to a dead Datanode is not available to HDFS any more. Datanode death may cause the replication factor of some blocks to fall below their specified value. The Namenode constantly tracks which blocks need to be replicated and initiates replication whenever necessary. The necessity for re-replication may arise due to many reasons: a Datanode may become unavailable, a replica may become corrupted, a hard disk on a Datanode may fail, or the replication factor of a file may be increased.

Cluster Rebalancing

The HDFS architecture is compatible with data rebalancing schemes. A scheme might automatically move data from one Datanode to another if the free space on a Datanode falls below a certain threshold. In the event of a sudden high demand for a particular file, a scheme might dynamically create additional replicas and rebalance other data in the cluster. These types of data rebalancing schemes are not yet implemented.

Data Integrity

It is possible that a block of data fetched from a Datanode arrives corrupted. This corruption can occur because of faults in a storage device, network faults, or buggy software. The HDFS client software implements checksum checking on the contents of HDFS files. When a client creates an HDFS file, it computes a checksum of each block of the file and stores these checksums in a separate hidden file in the same HDFS namespace. When a client retrieves file contents it verifies that the data it received from each Datanode matches the checksum stored in the associated checksum file. If not, then the client can opt to retrieve that block from another Datanode that has a replica of that block.

Metadata Disk Failure

The FsImage and the EditLog are central data structures of HDFS. A corruption of these files can cause the HDFS instance to be non-functional. For this reason, the Namenode can be configured to support maintaining multiple copies of the FsImage and EditLog. Any update to either the FsImage or EditLog causes each of the FsImages and EditLogs to get updated synchronously. This synchronous updating of multiple copies of the FsImage and EditLog may degrade the rate of namespace transactions per second that a Namenode can support.

However, this degradation is acceptable because even though HDFS applications are very data intensive in nature, they are not metadata intensive. When a Namenode restarts, it selects the latest consistent FsImage and EditLog to use. The Namenode machine is a single point of failure for an HDFS cluster. If the Namenode machine fails, manual intervention is necessary. Currently, automatic restart and failover of the Namenode software to another machine is not supported.

COMPARATIVE ANALYSIS OF HDFS AND GFS

A GFS cluster consists of a single master and Multiple servers and is accessed by multiple clients[8] and HDFS has a master/slave architecture. HDFS cluster comprises of a solitary NameNode, a master server that deals with the file system namespace and manages access to files by users.

Properties	**GFS**	**HDFS**
Design Goals	• Goal is to support large files • Built based on the assumption that terabyte data sets will be distributed across thousands of disks attached to commodity computer nodes. • Used for data intensive computing. • Store data reliably, even when failures occur within chunk servers, master, or network partitions. • GFS is designed more for batch processing rather than interactive use by users.	• One of the main goals of HDFS is to support large files. • Built based on the assumption that terabyte data sets will be distributed across thousands of disks attached to commodity compute nodes. • Used for data intensive computing. • Store data reliably, even when failures occur within name nodes, data nodes, or network partitions. • HDFS is designed more for batch processing rather than interactive use by users.
Processes	Master and chunk server	Name node and Data node
File Management	• Files are organized hierarchically in directories and identified by path names. • GFS is exclusively for Google only.	• HDFS supports a traditional hierarchical file organization • HDFS also supports third-party file systems such as CloudStore and Amazon Simple Storage Service.

Scalability	• Cluster based architecture • The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts. • The largest cluster have over 1000 storage nodes, over 300 TB of disk storage, and are heavily accessed by hundreds of clients on distinct machines on a continuous basis.	• Cluster based architecture • Hadoop currently runs on clusters with thousands of nodes.
Protection	• Google have their own file system called GFS. With GFS, files are split up and stored in multiple pieces on multiple machines. Filenames are random. There are hundreds of thousands of files on a single disk, and all the data is obfuscated so that it is not human readable. The algorithms uses for obfuscation changes all the time.	• The HDFS implements a permission model for files and directories that shares much of the POSIX model. File or directory has separate permissions for the user that is the owner, for other users that are members of the group, and for all other users
Security	• Google has dozens of datacenters for redundancy. These datacenters are in undisclosed locations and most are unmarked for protection. • Access is allowed to authorized employees and vendors only. Some of the protections in place include: 24/7 guard coverage,	• HDFS security is based on the POSIX model of users and groups. • Currentl y is security is limited to simple file permissions. The identity of a client process is just whatever the host operating system says it is. Network authentication protocols like Kerberos for user authentication

	Electronic key access, Access logs, Closed circuit televisions, Alarms linked to guard stations, Internal and external patrols, Dual utility power feeds and Backup power UPS and generators.	and encryption of data transfers are yet not supported.
Database Files	Bigtable is the database used by GFS. Bigtable is a proprietary distributed database of Google Inc.	HBase[15] provides Bigtable (Google) [16]-like capabilities on top of Hadoop Core.
File Serving	A file in GFS is comprised of fixed sized chunks. The size of chunk is 64MB. Parts of a file can be stored on different nodes in a cluster satisfying the concepts load balancing and storage management.	HDFS is divided into large blocks for storage and access, typically 64MB in size. Portions of the file can be stored on different cluster nodes, balancing storage resources and demand
Cache Management	• Clients do cache metadata. Neither the sever nor the client caches the file data. • Chunks are stored as local files in a Linux system. So, Linux buffer cache already keeps frequently accessed data in memory. Therefore chunk servers need not cache file data.	• HDFS uses distributed cache It is a facility provided by Mapreduce framework to cache application-specific, large, read-only files (text, archives, jars and so on) • Private (belonging to one user) and Public (belonging to all the user of the same node) Distributed Cache Files.
Cache Consistency	• Append-once-read-many model is adapted by Google. It avoids the locking mechanism of files for writing in distributed environment is avoided. • Client can append the data to the existing file.	• HDFS's write-once-read-many model that relaxes concurrency control requirements, simplifies data coherency, and enables high throughput access. • Client can only append to existing files
Communication	TCP connections are used for communication. Pipelining is used for data	RPC based protocol on top of TCP/IP

	transfer over TCP connections	
Replication Strategy	• Chunk replicas are spread across the racks. Master automatically replicates the chunks. • A user can specify the number of replicas to be maintained. • The master re-replicates a chunk replica as soon as the number of available replicas falls below a user-specified number.	• Automatic replication system. Rack based system. By default two copies of each block are stored by different Data Nodes in the same rack and a third copy is stored on a Data Node in a different rack (for greater reliability). • An application can specify the number of replicas of a file that should be maintained by HDFS. Replication pipelining in case of write operations.
Available Implementation	GFS is a proprietary distributed file system developed by Google for its own use.	Yahoo, Facebook, IBM etc. are based on HDFS.

V. CONCLUSION

Hadoop Distributed File System and Mapreduce are the components of Hadoop project owned by Apache. Google File System is a proprietary distributed file system and is exclusive for Google Inc. Mapreduce is the programming frame work used by Google.

VI. REFERENCES

[1]. Ke-Tha Yao,Robert F. Lucas et al. "Data anlysis for Massively distributed Simulations," Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2009

[2]. Vidyasagar S. D"A Study on Role of Hadoop in Information Technology era" Volume : 2 | Issue : 2 | Feb 2013 • ISSN No 2277 - 8160

[3]. Jiazhen Han, Zhengheng Yuan et al. "An Adaptive Scheduling Algorithm for Heterogeneous Hadoop Systems" 2017 IEEE ICIS 2017, May 24-26, 2017, Wuhan, China

[4]. Junbo Zhang, Dong Xiang, Tianrui Li, and Yi Pan "M2M: A Simple Matlab-to-MapReduce Translator for Cloud Computing" TSINGHUA SCIENCE AND TECHNOLOGY ISSNl1007-0214 01/12 pp 1-9 Volume 18, Number 1, February 2013

[5]. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[6]. https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/

[7]. http://www.aosabook.org/en/hdfs.html

[8]. R.Vijayakumari, R.Kirankumar, K.Gangadhara Rao "Comparative analysis of Google File System and Hadoop Distributed File System" International Journal of Advanced Trends in Computer Science and Engineering, Vol. 3 , No.1, Pages :553-558 (2014)