Towards Better Fault Tolerance in HDFS – A Survey Paper

Pranav Nair, Jignesh Vania
LJ Institute of Engineering and Technology
Ahmedabad, Gujarat, India

***Abstract*- Big data is a term that describes the sudden exponential growth and global generation of data that can be structured and unstructured. In simpler terms, big data can be thought of as a large dataset containing a variety of data types.**

I. INTRODUCION

When we consider the ambit of big data, it can be encapsulated into these 5 points.[14]

Variability: The amount of data is always going to increase exponentially. However another factor to consider is peak loads. Data generation and flow can vary with time and is very inconsistent. Along with the complexities involved with unstructured data, peak load management is very challenging.

Complexity: The source of data is very large. Related data can be sourced from multiple entities simultaneously. It is a gigantic task to correlate the data and preserving their relationships. Also, it is essential to cultivate and preserve hierarchies and data links to efficiently control data analytics.

Variety: With big data, the problem is not just about quantity. Applications deal with structured and traditional databases as well as unstructured text documents, image, videos and other compressed/uncompressed formats.

Velocity: Data is being streamed at previously unheard-of speeds. The biggest contributors to this are social media platforms and other utility applications such as RFID and sensors. Applications need to react quickly to manage such data sets.

Volume: There is a large volume of data being generated. In the past the biggest concerns related to volume was data storage. But with cheap and inexpensive hardware, that problem has been mitigated. Now, the focus has shifted to generating relevant value from this huge volume of data using analytics.

One might think about the relevance of this large unstructured dataset. Realistically speaking, this huge amount of data can be used to uncover trends in the global market, customer preferences, hidden patterns – weather, agriculture etc., detect previously hidden patterns and other unknown correlations. The market term used for that is called big data analytics.

Big data analytics aims at arming a business with a plethora of information that will help the business in making well-informed decisions. It enables data scientists and data modellers to analyze large volumes of data including clickstream data, IP traffic, web logs, content data from web pages, emails and even social media content.

As mentioned previously, semi-structured and unstructured data are not tailor-made for data warehouses based on relational databases. As a result, many organizations banking on data analytics have turned to a new class of technologies including Hadoop and its dependent tools like YARN, MapReduce, Hive, Zookeeper etc[7].

II. HADOOP

Hadoop is an open source software project developed by Apache that facilitates distributed processing of large data sets[12]. However the main feature is that Hadoop lets a user do this on a platform built up of clusters of commodity servers. It allows the user to scale up from a single server to thousands of machines. It also provides a very high degree of fault tolerance. Hadoop does not look to sophisticated hardware solutions for providing this much elasticity. It banks on the software's ability at application layer to cope with breakdown[1]. The projects in Apache Hadoop include[12]

Hadoop Common

Hadoop Distributed File System
Hadoop YARN (Yet Another Resource Negotiator)
Hadoop MapReduce

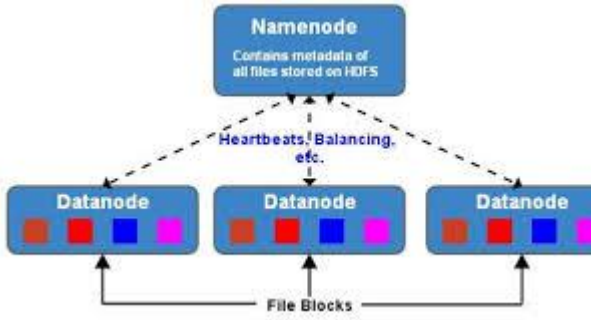
Hadoop Distributed File System

HDFS is a distributed file system built to run on commodity servers. The overall design of HDFS is very similar to the existing file systems. However, upon closer look, one can find many significant dissimilarities.HDFS is designed to be highly fault tolerant. It should easily be deployed on cheap,

inexpensive hardware. It also provides high throughput access to application data. As can be seen, HDFS is optimal for applications that have large data sets[8].

HDFS Architecture

HDFS implements a master/slave architecture[11]. Each cluster has a single master called NameNode along with multiple slaves called DataNode. So each Hadoop cluster is governed by a NameNode and it has control over tens or thousands of DataNodes. The NameNode manages the file system and regulates access of files by clients. All client requests need to fly by the NameNode. The DataNodes are attached to the NameNode and manage file storage on the machines they run on. HDFS splits the user data into blocks and these blocks are stored in a set of DataNodes. Again, the NameNode executes all operations on the file system namespace like opening, closing and renaming files and directories. It also manages the mapping of files to blocks on the DataNodes. Tasks such as replication and block creation are done by the NameNode upon instructions of the DataNode.



HDFS Architecture[16]

In short, the Namenode stores all kinds of metadata like namespace of the distributed file system, file-block mappings, block allocations, access regulations, file descriptions and so on. Whenever a HDFS client wants to access a file, it has to contact the Namenode[7][8].

As can be seen, the Namenode stores a lot of metadata information that is crucial to working of the Namenode. So, the Namenode can be classified as a Single Point Of Failure. If and when the Namenode goes offline, the entire cluster goes down. And there is no point of hosting all this huge amount of data if you cannot gather required information from it within a required amount of time.

The entire platform of Hadoop is based on the notion of High Availability.Let us look at some factors that lead to the requirement of Namenode's High availability.[11]

- The data structure used by Namenode is inode, which contains namespace information, permissions and access times.
- It manages to file-to-block conversion and vice-versa.
- The Namenode maintains the hierarchy of namespace tree.

The namespace and the inodes collectively form what is known as fsimage. This fsimage is entirely stored in the Namenode machine's Random Access Memory. The *image* is persisted in the Namenode machines local file system and is referred to as checkpoint.

- All the transactions/changes that occur in HDFS are recorded in a log called journal. This journal is a write-ahead file. Whenever a transaction is successfully completed, the record is flushed from the journal before sending an acknowledgement to the client[13].
- Multiple checkpoints and journals are kept for use in case of node failure. So each successful transaction is signalled by a flush-sync operation.
- The Namenode is a multi-threaded system and is capable of serving multiple client requests simultaneously, which creates a bottleneck during the flush-sync process.

During start up, the Namenode must complete the following actions.[9]

1. Read the edit logs.
2. Compare the logs with fsimage file to check for any new operations that need to be persisted.
3. Update the fsimage file with the new actions.

These operations can take a long time to complete depending upon some uncertainties. They include the following factors.[1][2]

1. Very large edit logs
2. Degraded storage disks can slow down reading fsimage and edit logs, as well as writing new checkpoints.

However, HDFS does not have a very high magnitude of High Availability[5][6]. Clusters are bound to go offline at some point of time. However, there are some solutions put forward to tackle this problem. Let us have a look at them.

III. SECONDARY NAMENODE

Whenever any modifications are done to the file system, it is stored as a log in the native file system of the Namenode machine. This log file named called editlog. Whenever a Namenode is started, it looks at an image of the HDFS state called fsimage. This fsimage file contains the updated instance of the file system. It then has to update the fsimage file. To do that it starts going though each entry in the editlog and then applies the edits to the file system. Then the updated fsimage is written to the file system state after which normal operation resumes. After this, the editlog data is deleted.[10]

The Namenode updates the fsimage file with data from editlogs only during Namenode start up. However there is no protocol regarding periodic start up of the Namenode. So, the schedule for doing that might be haphazard. Because of this, it is possible that the size of editlog could get very large. Because of this, the Namenode might take a long time to start up after failover[3].

This is where the Secondary Namenode comes to play. It periodically updates the fimage file with editlog. It oversees the size of editlog and does not allow it to go beyond a pre-defined limit.

The Secondary Namenode is run on a different machine than the primary Namenode. That is because its requirements are similar to that of the primary Namenode.[10]

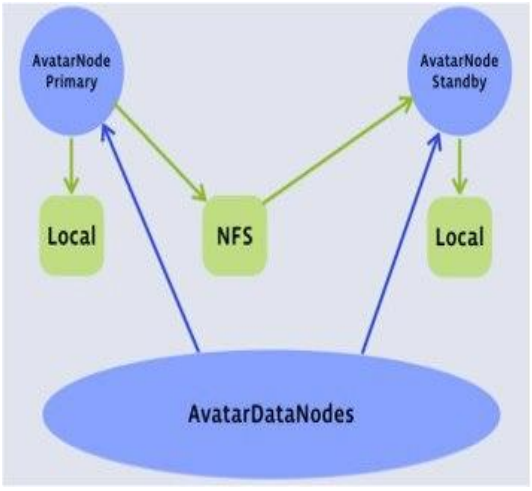
2 parameters exist, that control the process of file synchronization taken up by Secondary Namenode.

- *fs.checkpoint.period*: it is set to 1 hour. After which the process has to execute.
- *fs.checkpoint.size*: set to 64MB by default. Defines maximum size limit of editlog file.

The secondary Namenode stores this checkpoint in a directory. As said earlier the directory structure of Secondary Namenode is exactly the same as Primary Namenode. So there is no need for any of type-conversion or compatibility check to be done before the checkpoint is loaded into the Primary Namenode in case of any failure.

IV. AVATARNODE

The Avatarnode lets go of the Namenode and brings an architecture that consists of a Standby Avatarnode and Primary Avatarnode. This implementation was developed by Facebook and it has contributed back to the community as open source application[17].



Avatarnode Architecture[17]

Their aim is to offer a highly available system, where the Namenode is equipped with with hot failover and failback.

The Avatarnode is a two-node, highly available Namenode. It provides additional fault tolerance by implementing a manual failover approach. The fundamental concepts in Avatarnode are:

1)There are 2 Namenodes – a Primary AvatarNode and a Standby Avatarnode. Either node can adapt a new Avatar, based on necessity.

2) Zookeeper maintains the status of the current Primary.

3) A modified Datanode sends block reports to the Avatarnodes.

4) Before beginning each transaction, and again in the middle of a transaction, a modified HDFS client checks Zookeeper to see if it fails. Because of this, writes can be successfully completed even if a Avatarnode failover takes place in the middle of a write.

The paper by Feng Wang, Jie Qiu, Jie Yang,. Bo Dong, Xinhui Li, Ying Li., titled" Hadoop High Availability through Metadata Replication"[4] analyzes the problem of Namenode being SPOF. To solve this, a metadata replication strategy is proposed. There are 3 phases: initialization phase, replication phase and failover phase. In the initialization phase, the standby Namenodes need to register with the primary Namenode, effectively implementing a pub-sub pattern. The main task is to synchronize the metadata contained in the primary Namenode with all other slave nodes.In replication phase, the metadata is replicated at runtime. This is crucial as there all multiple locations for synchronization of metadata.The failover phase is the phase where the manual takeover occurs. When the primary Namenode

fails, one of the standby nodes becomes the new primary and takes over all communications.

A paper titled "NCluster: Using Multiple Active Name Nodes to Achieve High Availability for HDFS,"[3] written by Zhanye Wang; Dongsheng Wang also tries to address the issue of Namenode failure. This approach uses metadata replication to ensure that the Namenode remains available to serve the cluster.Here the solitary Namenode is replaced with by Namenode cluster. Depending on the cluster size, 3-5 Namenodes are set up. Moreover, the strategy used to manage this cluster is novel. Here, the authors use a single writer, multiple readers strategy. A single node is called primary Namenode. It handles all write requests and some of the read requests. The rest of the Namenodes are called hot standbys. These nodes are read only. The primary replicates all the metadata at uptime to the hot standby Namenodes. They implement a pub-sub system to handle this process.

V. CONCLUSION

Hadoop is a platform that is tailor-made for the data requirements of the future. It provides unstructured data storage and is very efficient for big data storage and manipulation. It uses various projects like YARN, MapReduce, HDFS, HBase, Hive and Zookeeper.

Even though Hadoop has many features that support fault tolerance, it is still susceptible to problems arising from hardware failures. Analysis has shown us that HDFS has a Single Point Of Failure. And that SPOF is the master node of the cluster called Namenode. The Namenode is responsible for storing metadata of the cluster including replication factor, ownership, access information etc. It is also responsible for converting the files into blocks and storing them in distributed Datanodes. Within the Datanodes, the Namenode needs to store data onto different racks. All these characteristics contribute to state the obvious fact that Namenode is the main element in HDFS that provides fault tolerance.

We saw some approaches that deal with this problem. One approach uses a secondary Namenode. It periodically collects the metadata information that is created by the Namenode and backs it up. However, this is nothing but a cold standby. The secondary/backup namenode cannot serve client requests.

The next solution we discussed is Avatarnode. This implementation also proposes a dual standby system, where either nodes can act as primary or secondary. Metadata synchronization between these nodes is done using Zookeeper, which is a synchronization service project also developed by Apache. However, it should be noted that Zookeeper itself has some issues. The process of metadata synchronization is inefficient. It generates a lot of disk I/O because of frequent updates amongst the namenodes. This leads to delay. We can use SSDs to speed up the I/O but SSDs can be another point of failure in that case.

All other approaches discussed above follow the tactic of metadata replication. However, they have some or the other issues that are not desirable or need to be removed. The approach taken by Feng Wang, Jie Qiu, Jie Yang,. Bo Dong, Xinhui Li, Ying Li[4]implements a cluster of namenodes. They divide the entire process into 3 phases; initialization, replication, and failover. The process of failover is debatable as it uses a dynamic approach to elect the new namenode upon failure of the primary. It is time consuming process and generates considerable overhead.

The last approach discussed[3] again implements a cluster of namenodes. They develop a design where there are 3-5 active namenodes in the cluster. The primary namenode can read from and write to the file syste. However, the backup namenodes can serve read requests of the clients. Thus they are not cold standby namenodes as designed in some systems. However, with the amount of nodes in the cluster, synchronization between the namenodes in cluster can generate a large overhead. This is undesirable in the case of legacy data.

So one can see that there is a need for improvement in the problem of fault tolerance of HDFS. The current solutions all lack in some way or other and a better approach is required to ensure that the file system is more fault tolerant and is highly available to the system.

REFERENCES

[1] Dwivedi, K.; Dubey, S.K., "Analytical review on Hadoop Distributed file system," *Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference* - , vol., no., pp.174,181, 25-26 Sept. 2014

[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] Zhanye Wang; Dongsheng Wang, "NCluster: Using Multiple Active Name Nodes to Achieve High Availability for HDFS," *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on* , vol., no., pp.2291,2297, 13-15 Nov. 2013
[4] Feng Wang, Jie Qiu, Jie Yang,. Bo Dong, Xinhui Li, Ying Li.," Hadoop High Availability through Metadata Replication" 2009 IBM Corporation.
[5] Kai Fan; Dayang Zhang; Hui Li; Yintang Yang, "An Adaptive Feedback Load Balancing Algorithm in HDFS," *Intelligent Networking and Collaborative Systems (INCoS), 2013 5th International Conference on* , vol., no., pp.23,29, 9-11 Sept. 2013
[6] S. Ghemawat, H. Gobioff, and S.T. Leung. The Google File System. SOSP 03: proceedings of the 19th ACM Symposium on Operating Systems Principles
[7] E. Baldeschwieler and D. Cutting, State of Hadoop. Hadoop Summit 2009.
[8] http://www.aosabook.org/en/hdfs.html
[9]http://hortonworks.com/blog/ understanding-Namenode-startup-operations-in-hdfs
[10] http://wiki.apache.org/hadoop/Namenode
[11]http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
[12] http://hadoop.apache.org
[13] http://stackoverflow.com
[14] http://www.sas.com/en_us/insights/big-data/what-is-big-data.html
[15]T. White. Hadoop: The Definitive Guide. O'Reilly Media 2009.
[16] blog.spec-india.com/hadoop-core-components
[17]https://gigaom.com/2012/06/13/how-facebook-keeps-100-petabytes-of-hadoop-data-online/