International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552, Volume-3, Issue-2, MARCH-2015

Improved Fault-tolerant Architecture for HDFS using Distributed Namespace

Veena Dange, Pallavi Deshmukh, Sayali Deshpande, Madhubala Girase, Prof. Ratan Deokar

Abstract- **In HDFS, the Namenode handles all the metadata about the files. Client contacts the Namenode for accessing any file to get its mapping to the blocks on Datanodes. Namenode has to store this data in its RAM to allow faster access to clients on their requests. Hence HDFS Datanodes' metadata is restricted by the capacity of the RAM. This makes the Namenode as the single point of failure in HDFS. In our approach we present multiple Namenodes with distributed namespace using Chord protocol. Multiple Namenodes will be arranged in Chord ring to provide scalable and fault tolerant architecture in HDFS.**

Keywords—Chord DHT, HDFS, MLT, Namenode.

I. INTRODUCTION

Hadoop is ideal for storing large amounts of data, like terabytes and petabytes and uses a distributed file system for data storage called Hadoop Distributed File System (HDFS). There are thousands of server machines in hadoop cluster. This implies more possibility of hardware failure. Thus, fault detection and automatic prompt server recovery are fundamental architectural goals of HDFS. The same applies for the Namenode server, as it is the only master component that provides access to entire HDFS cluster. HDFS's performance heavily relies on the availability of single Namenode machine.

Hadoop applications run over HDFS, which has a single Namenode for storage of namespace. This entire namespace is maintained in Namenode's RAM so that metadata can be fetched at faster rate. Hence, HDFS Datanodes' metadata is restricted by the capacity of the RAM of Namenode. According to statistics on YAHOO clusters, a file on average consists of 1.5blocks and as Namenode uses less than 200 bytes to store a single metadata object, it takes 600 bytes to store an average file in Namenode's RAM. To store 100 million files (referencing 200 million blocks), a Namenode should have at least 60GB of RAM[12].

Manuscript received March 04, 2015.
Veena Dange, Computer Department, Pimpri Chinchwad College Of Engineering , Pune, India.
Pallavi Deshmukh, Computer Department, Pimpri Chinchwad College Of Engineering , Pune, India.
Sayali Deshpande, Computer Department, Pimpri Chinchwad College Of Engineering , Pune, India.
Madhubala Girase, Computer Department, Pimpri Chinchwad College Of Engineering , Pune, India.

Chord's main goal is the location of entities in P2P environments, like documents, files, or any resource that one might want to share in a computer network. It is a distributed lookup protocol which maps a given key onto a node. Data is easily placed in a Chord by associating a key with each resource item. Along with Chord protocol's functionality, we also make use of Metadata Lookup Table for faster lookup.

High Availability Issue: An HDFS needs a single controller server machine, the Namenode. This becomes a single point-of-failure for an HDFS implementation. If this Namenode fails to work, the entire system comes to a halt-state. After it gets back online, it must respond to all client requests and Datanode manage operations. The Namenode server restoration process can take over half an hour for a large cluster. The HDFS also includes a Secondary Namenode, which should not be thought of as the replacement for the Primary Namenode server. In case of primary Namenode failure, it is not the responsibility of the secondary Namenode to take over the primary. In reality, it only functions to build the periodic image-based snapshots of the Primary Namenode's directory information and save them to local/remote directories. These image-based checkpoints can only be used to restart a failed Primary Namenode without having to replay the entire journal of HDFS actions, the edit log to create an up-to-date directory structure.

II. RELATED WORK

The high availability issue of HDFS is addressed by making use of various strategies. Few of them are as follows,

1. Hot Standby: In this technique, a hot standby server node is maintained with complete and up-to-date copy of the state of its primary node. In case of primary Namenode failover, it can be replaced by the standby Namenode server within a short period of time. The backup node of Hadoop can be used to provide the high available solution.

The use of hot standby server node does not ensure the distribution of namespace. Thus, even though it takes up the primary's responsibility, it does not achieve scalability of the cluster due to the RAM size.

2. HDFS Federation: Apache addressed the issue of Namenode scalability and proposed the solution with multiple Namenodes. In a single cluster, multiple independent Namenodes are configured managing their own namespace volumes. The datanodes register themselves with all federated Namenodes. A block pool is a set of blocks that belongs to a single namespace. Datanodes store

blocks for all the block pools in the cluster. It is managed independently of the other block pools. This allows a namespace to generate block ids for new blocks without the need for coordination with other namespaces. A client-side mount table is maintained to provide a global view of the namespaces.

As it makes use of static subtree partitioning, the workload may not be evenly distributed among Namenodes.

This paper proposes a fault tolerant, highly available and widely scalable HDFS architecture having multiple Namenodes, with distributed namespace. The Chord protocol is used for namespace distribution amongst Namenodes.

III. EXISTING HADOOP ARCHITECTURE

HDFS has master/slave architecture. A typical Hadoop cluster is mainly comprised of a Namenode
and several Datanode machines as shown in Figure 1.
The Namenode manages the HDFS namespace and regulates access to files that are requested by clients. Datanodes, which manage storage attached to the nodes that they run on, store the actual data.

The Namenode and Datanode are software programs designed to run on everyday use machines. HDFS can be run on any machine that supports Java and therefore can run either a Namenode or the Datanode software. Usage of the highly portable and all pervasive Java language means that HDFS can be deployed on a wide range of machines. A typical deployment has a dedicated machine that runs only the Namenode software. Each of the other machines in the cluster runs one instance of the Datanode software. The architecture does not prevent running multiple Datanodes on the same machine but, in practice, that is rarely the case.

MapReduce: For huge data sets of distributed applications, MapReduce is well known for its simplicity and functionality. It allows moving the computation to the data itself. Hence, reducing the cost of data migration. It serves as an integral part of Hadoop to support distributed computing on large data sets on clusters of computers. MapReduce can be applied on the data stored in either a file system (unstructured) or within a database (structured). During a typical Map function, the master node accepts a major input, slices it into several minor sub-problems, and allocates them to worker nodes. A worker node could repeat this process again, if needed, resulting in a multi-level tree structure. Finally, the worker node processes the received problem chunk, and returns the processed data back to its master node. In the "Reduce" function, the master node receives the processed sub-problems and aggregates them in some way to form the output.

MapReduce has been wisely chosen to be the part of Hadoop project because it enables unnoticed distributed processing of the map and reduction operations. Hence, multiple map functions can be run in parallel, given that, each mapping operation is autonomous of the other. In reality, however, this condition is limited by the data source and/or the number of CPUs near that data. Likewise, a set of 'reducers' can be run all at the same time during the reduction phase, given that all outputs of the map operation, which share the same key, are presented to the same reducer simultaneously. Although, this procedure may look ineffective compared to other sequential algorithms, MapReduce can be functional to potentially larger datasets than "commodity" servers can handle. Hence, for instance, using MapReduce, a large server cluster can sort a petabyte of data in only a few hours. Moreover, this parallelism also enables a probability of high availability in case of a partial failure of servers or storage during the operation. That is, if one mapper or reducer fails, the work can be rescheduled, given that the input data is still available.

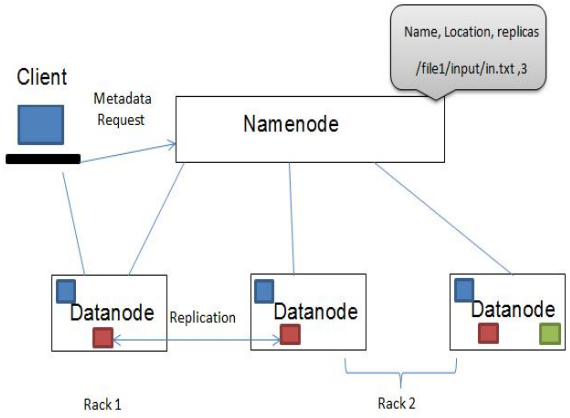

Fig 1. Hadoop Architecture

IV. CHORD PROTOCOL

The function of Chord protocol is primitive: for a unique key, it maps the key to a node. This node, depending on the application using Chord, could be in charge for storing a corresponding value for its key. Chord employs consistent hashing to allocate keys to Chord nodes. Because each node receives approximately the equal number of keys, consistent hashing performs load balancing and needs comparatively less reallocation of keys when nodes join and leave the system.

The Hash function assigns each node and key, an m-bit identifier using a base hash function SHA-1. The node identifier is calculated by performing the hashing on the IP address of the node. The identifier for key can be generated by hashing any attribute of the file.

ID(node)=hash(IP)

ID(key)=hash(key)

In an m-bit identifier space, there are 2^m identifiers. Identifiers are ordered on an identifier circle modulo 2^m [4] . The identifier ring is called Chord ring. Key k is assigned to the first node whose identifier is equal to or follows (the identifier of) k in the identifier space. This node is the successor node of key k.

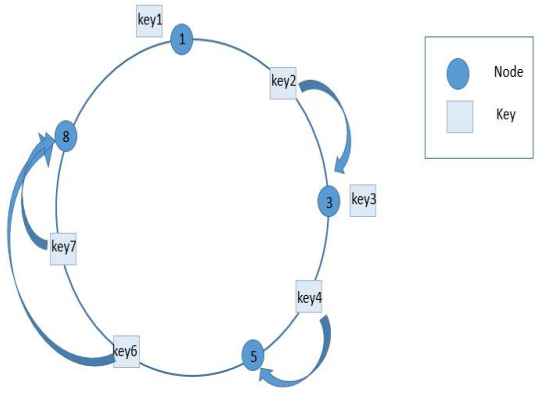

Fig 2. Chord Ring

As shown in fig.2, there are four nodes in the system with identifiers 1,3,5,8 respectively, arranged in clockwise ring. There are total six keys 1, 2, 3, 4, 6, 7. Key 1 and key 3 is assigned to node 1 and node 3 respectively. As there is no node with identifier 2, key 2 is assigned to first node with identifier higher than 2 which is node 3. Similarly key 4 is assigned to node 5 and key 6, key 7 to node 8.

V. THE PROPOSED ARCHITECTURE

The management of ever increasing size of the namespace which holds data related to billions of files comes across as a great challenge. It imposes a threat to high scalability and performance of metadata services. An approach to handle the mentioned threats could be the use of distributed namespaces. This can be done by multiple Namenodes instead of the centralized Namenode. The proposed architecture addresses the issues of high scalability, SPOF (Single Point of failure), high availability, load balancing without compromising the performance.

Metadata Lookup Table

The metadata lookup table is maintained at the client side. The entries in the MLT consist of the range of hash values and the IP address of Namenode responsible for that range. Whenever the client access the file, the file ID is searched into the entries of the MLT and the IP address of the Namenode responsible for that file is obtained. MLT introduces an additional level of indirection between the client and the Namenode. We make use of the MLT structure instead of Chord Protocol's Finger table to provide faster access to metadata.

Namespace Distribution by Hashing

In our approach the Each Namenode is assigned an identifier. This identifier is actually the SHA-1 hash of IP address of respective Namenode. Each Namenode is responsible for a set of keys which falls between its predecessor and itself. The file identifier is generated by hashing the path of file. File k is assigned to the first node whose identifier is equal to or greater than k in the identifier space, regardless of the owner of the resource that generated this key. This Namenode is called the successor node of k. The Metadata Lookup Table is the structure implemented at client side which helps to directly identify the responsible Namenode for query.

When client want to perform any operation it will calculate SHA-1 hash of file path. Then it will look up in the MLT for the range in which the file identifier sits, which will ultimately give the IP address of Namenode responsible for that request. Now, client can directly contact that Namenode for operation.

Though it looks like multiple Namenodes make HDFS complex, the single point-of-failure HDFS Namenode and its RAM limitation to keep all the files metadata, stored in the Datanodes, required an alternative solution. Chord integration into HDFS Namenode provides a reliable and efficient solution to this problem.

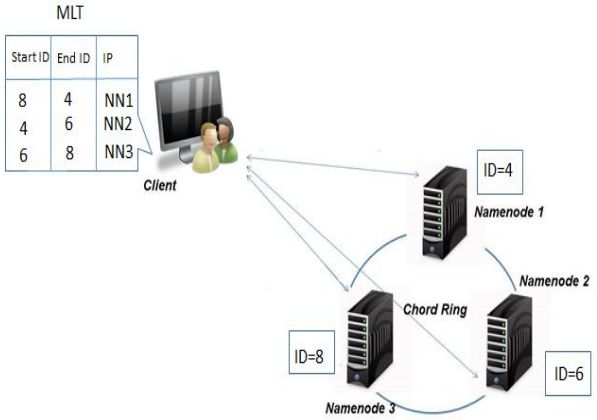

Fig 3. Proposed architecture

VI. CONCLUSION

In this paper, a new fault-tolerant architecture for HDFS is introduced. The dependency of whole system on a single Namenode server is decentralized using distribution of namespace on multiple Namenodes.

References

[1] Towards a scalable HDFS architecture, Farag Azzedin IEEE 2013.

[2] Load rebalancing for Distributed File System in Clouds, IEEE transactions on Parallel and distributed system.

[3] ALDM: Adaptive Loading Data Migration in Distributed File System, IEEE transactions 2013.

[4] Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications.

[5] Classification based Metadata Management for HDFS, 2012 IEEE 14th International Conference on High Performance Computing and Communications.

[6] A Novel Blocks Placement Strategy for Hadoop, 2012 IEEE/ACIS 11th International Conference on Computer and Information Science.

[7] https://hadoop.apache.org/docs/r2.3.0/hadoop-project-dist/hadoop-hdfs/Federation.html

[8] From Backup to Hot Standby: High Availability for HDFS,2012 31st International Symposium on Reliable Distributed Systems

[9] An Adaptive Feedback Load Balancing Algorithm in HDFS, 2013 5th International Conference on Intelligent Networking and Collaborative Systems

[10] NCluster: Using Multiple Active Namenodes toAchieve High Availability for HDFS , 2013 IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing

[11] http://www.ibm.com/software/data/infosphere/hadoop/

[12] https://developer.yahoo.com/blogs/hadoop/scalability-hadoop-distributed-file-system-452.html

VEENA DANGE
Bachelor of Computer Engineering, student of Pimpri Chinchwad College Of Engineering, Pune.

PALLAVI DESHMUKH
Bachelor of Computer Engineering, student of Pimpri Chinchwad College Of Engineering, Pune.

SAYALI DESHPANDE
Bachelor of Computer Engineering, student of Pimpri Chinchwad College Of Engineering, Pune.

MADHUBALA GIRASE
Bachelor of Computer Engineering, student of Pimpri Chinchwad College Of Engineering, Pune.

PROF. RATAN DEOKAR
Is a lecturer at Pimpri Chinchwad College of Engineering, Pune, in the Computer Department . He has bachelor's and master's degree in IT engineering. He has 7 years of teaching experience.