IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 3, Ver. II (May-Jun. 2014), PP 58-62
www.iosrjournals.org

Highly Available Hadoop Name Node Architecture-Using Replicas of Name Node with Time Synchronization among Replicas

Soubhagya V N[1], Nikhila T Bhuvan[2]
[1](Department of IT, Rajagiri School of Engineering and Technology/MG university, India)
[2](Department of IT, Rajagiri School of Engineering and Technology/MG university, India)

Abstract : *Hadoop is a Java software framework that supports data - intensive distributed applications and is developed under open source license. It enables applications to work with thousands of nodes and petabytes of data. The two major pieces of Hadoop are HDFS and MapReduce. HDFS works with two types of hardware machines, the DataNode (Slave machine) which is the machine on which application's data is stored and the NameNode (Master machine) which store the metadata of file system. Where NameNode is the only single machine for storing metadata of file system and is the Single Point of Failure (SPOF) for the HDFS. SPOF of NameNode machine affects the overall availability of Hadoop. When NameNode goes down the entire system become offine and cannot do any operation until NameNode gets restart. If the NameNode machine fails, the system needs to be re-started manually, making the system less available. This paper proposes a highly available architecture and its working principle for the HDFS NameNode against its SPOF utilizing well-known 2-Phase Commit (2PC) Protocol and election by bully with Time synchronization mechanism.*

Keywords - *Hadoop, HDFS, Two Phase Commit Protocol, Berkeley algorithm for time synchronization, Name node.*

I. Introduction

Now days there are so many organizations in the world that owns very large amount of data on the internet. To store and process such large amount of data, require special hardware machines, which is not a good deal in terms of money, as well as the processing capability of single machine cannot scale to process such huge data on it. Hadoop is an open source frame work used for processing such very large data sets. Hadoop mainly do this job with the help of MapReduce and Hadoop Distributed file system (HDFS). Hadoop provides a distributed file system and a framework for the analysis and transformation of very large data sets using the MapReduce paradigm. An important characteristic of Hadoop is the partitioning of data and computation across many (thousands) of hosts, and executing application computations in parallel close to their data. A Hadoop cluster scales computation capacity, storage capacity and IO bandwidth by simply adding commodity servers.

Apache Hadoop consists of two primary components: HDFS and MapReduce. HDFS, the Hadoop Distributed File System, is the primary storage system of Hadoop, and is responsible for storing and serving all data stored in Hadoop. MapReduce is a distributed processing framework designed to operate on data stored in HDFS. HDFS has long been considered a highly reliable file system. An empirical study done at Yahoo! concluded that across Yahoo!'s 20,000 nodes running Apache Hadoop in 10 different clusters in 2009, HDFS lost only 650 blocks out of 329 million total blocks. The vast majority of these lost blocks were due to a handful of bugs which have long since been fixed. Despite this very high level of reliability, HDFS has always had a well-known single point of failure which impacts HDFS's availability.

HDFS is the file system component of Hadoop.HDFS is doing its part of work very efficiently in Hadoop since it was developed. HDFS stores file system metadata and application data separately. As in other distributed file systems, like PVFS, Lustre and GFS, HDFS stores metadata on a dedicated server, called the NameNode. Application data are stored on other servers called DataNodes. All servers are fully connected and communicate with each other using TCP-based protocols. So HDFS works with two types of hardware machines, the DataNode (Slave machine) which is the machine on which application's data is stored and the NameNode (Master machine) which store the metadata of file system. Application's data is stored on DataNode using small blocks of data with usually three replicas. Where NameNode is the only single machine for storing metadata of file system and is the Single Point of Failure (SPOF) for the HDFS. SPOF of NameNode machine affects the overall availability of Hadoop. When NameNode goes down the entire system become offline and cannot do any operation until NameNode gets restart. If the NameNode machine fails, the system needs to be re-started manually, making the system less available.

II. Proposed Architecture

We have thoroughly studied and analyzed the architecture of HDFS with focus on its NameNode's SPOF problem. To eliminate the problem of HDFS NameNode, we have devised architecture of HDFS and its working principle. This architecture and working principle eliminates SPOF of NameNode by initially replicating NameNode at separate machines (in our case two machines) only once and then starts working normally with its DataNodes.

The core idea of this architecture is that once NameNode get replicated after that, it will do all operation on its metadata by using well known 2-Phase Commit (2PC) Protocol with some variations. Initially the NameNode will be acting as Coordinator NameNode, and its replicas will be Participants NameNode machines. The Coordinator and Participants machines will perform any update to their metadata according to 2PC protocol. To increase the throughput of update operations we have made some variations in 2PC according to our needs. Due to 2PC protocol both of the replicas of Coordinator NameNode will be fully updated all time, hence in case Coordinator NameNode fails or even completely goes down then any of its replicas can become Coordinator and hosts the DataNodes. The new Coordinator can be elected using Election by Bully algorithm.

2.1 NameNode Replication

To increase the availability of any system, the very simple technique that can be used is redundancy. Here at initial step, we have done the same thing by replicating NameNode at two different machines that are called Participant NameNodes. In our case we have taken replica factor two, but it can be varied according to degree of availability requirement. Hence when the NameNode starts for the first time, it will be replicated only once on Participant NameNode machines. Once replication is complete, the system will then work under the principle of Two Phase Commit Protocol with some modifications.

2.2 Two Phase NameNode Commit (2PNNC) Protocol

In our proposed solution once NameNode is replicated (for once in its life), after that NameNode and its replicas will work under the principle of Two Phase Commit (2PC) Protocol. We have not exactly used the same 2PC protocol, but have made some variations in it according to our needs. Therefore we have renamed this protocol as Two Phase NameNode Commit (2PNNC) protocol. Here the NameNode will be called Coordinator NameNode and its replicas will be known as Participant NameNode machines. Every Participant NameNode is associated with an update-queue that will hold multiple vote-requests from the Coordinator. Any of the Participants can send a vote-commit to coordinator, as soon as it receives the vote-request and its update queue (discussed below) is empty. So, in this case the Coordinator need not wait for all of the vote-commit replies from the Participants, rather it will continue to send global-commit against a single vote-commit from any of the Participant. The remaining Participants can perform their commit operation later, but there must be at least one Participant who will update itself with the Coordinator at same time. After the vote-request from the Coordinator, if the update-queue is partially filled then the Participant can accept the vote-requests, but will send vote-commit to coordinator until it has performed all the pending operations in its update-queue. The Participant will send vote-abort in the case when its update-queue is full. The working principle of update-queue is explained in detail later in this Section. The 2PNNC protocol for our proposed solution is explained below:

- *Phase 1 a.* Coordinator NameNode sends vote request to all Participant NameNodes.
- *Phase* 1 b. When any of the Participant NameNode receives vote-request from Coordinator then it will either reply vote-commit or vote-abort. It will send vote- abort if its update-queue is full of previous vote-requests, otherwise it will keep the vote-request in its update-queue and sends vote-commit to coordinator when its update-queue become empty by performing all previous requested updates.
- *Phase 2* a. The Coordinator NameNode collects all votes from the Participants. If it gets vote-commit from any one of its Participants NameNode, it commits its update operation and sends out global-commit to all Participant NameNodes. If it receives vote-abort from any of its Participants NameNodes then it will abort its update operation and will send global-abort to all Participant NameNodes.
- *Phase* 2 b. Every Participants Name-node machines will wait for Coordinator global- commit and global-abort and will respond accordingly.

In this architecture every Participant NameNode will maintain a small length FIFO update-queue (see Figure 1) which will keep track of the incoming vote- request from Coordinator NameNode for updating some part of metadata. The main purpose for this update-queue is to respond to Coordinator quickly, so that the throughput of the system can be increased. The throughput is increased in such ways that, the Coordinator need not to wait for vote-commit replies from all of its Participants.

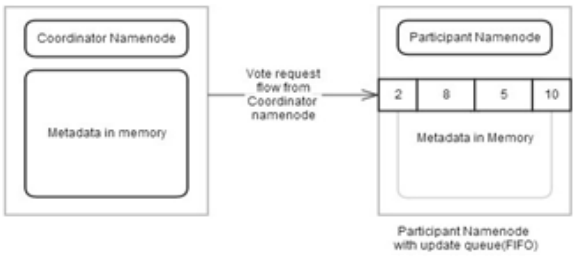

Fig. 1. Participant Name Nodes updation queue

Figure 2 and figure 3 gives very clear illustration of working principle of Coordinator NameNode and Participant NameNode respectively. Figure 2 and figure 3 gives very clear illustration of working principle of Coordinator NameNode and Participant NameNode respectively.

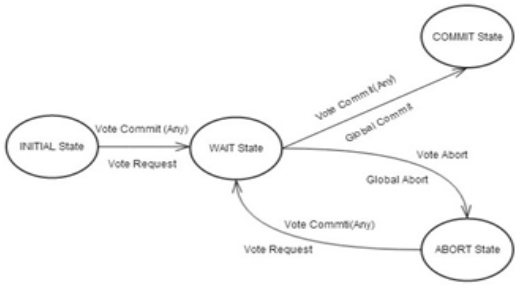

Fig. 2. Coordinator Name Nodes working principle

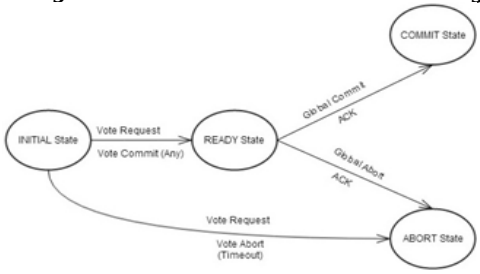

Fig. 3. Participant Name Node Working Principle

2.3 Failure of NameNodes in HDFS and Election Process

As we have described in previous section, that, Hadoop runs over low cost commodity hardware machines, where the probability of failure is higher. Our proposed architecture and working principle of Hadoop NameNode is fully equipped for dealing with such failures. Here in this architecture we can face two kinds of failures, the Coordinator and the Participant NameNode failure.

2.3.1 Participant NameNode Failure and Recovery

In the case of Participant NameNode failure, it will be noticed or detected by the Coordinator NameNode machine. This detection is done by Coordinator if it does not receive votecommit from any of the its Participant NameNode for long duration of time. After that it will send a heartbeat message to that particular Participant NameNode. If it does not replied to Coordinator in fixed amount of time, the Coordinator will declare this Participant NameNode as failed Participant and will generate alarm of Participant's failure. The failed Participant machine once recover and gets restart, then it sends a heartbeat reply to Coordinator. The Coordinator in response will assign it to another active Participant NameNode machine for replication and synchronization. Once it is being replicated and fully updated from its neighbouring active Participant, it can become a part of active Participant NameNode pool. The replication and updation was assigned to other active Participant to reduce the load over Coordinator NameNode machine

2.3.2 Coordinator NameNode Failure Detection and New Coordinator Election

On every vote-commit, the Participant NameNode will maintain a priority number with it. It will also have a serial number associated with it, which will be assigned to all Participants NameNodes at the time of becoming a member of the pool. This serial is number will not change in their course of execution, e.g in case of three Participant NameNode machines they will have serial numbers as 1, 2 and 3. However the priority number

of each Participant will be initialized as 0(zero) at the beginning, and will be incremented when they send vote-commit to the Coordinator. The Coordinator failure will be detected by any of the Participant NameNode if it gets no vote-request from Coordinator for some fixed amount of time, in response to this, it will send heart beat message to the Coordinator. If Coordinator does not reply to it, then that Participant NameNode machine will start election by Election Algorithm ”Election by Bully”. Below is the complete process illustration of Election by Bully with some variation in it according to our needs.

- If any of the Participant NameNode detects the Coordinator NameNode failure, it can start election process by sending an election message with its priority number to all Participant NameNode machines. Assuming that it does not know the priorities of other Participant NameNodes, but knows their serial numbers.

- If a Participant NameNode with high priority number gets an election message from other Participant NameNode machine with low priority number, then it will send take over message to that Participant NameNode and that Participant NameNode is out of the election.

- If the Participants does not get take-over message from any other Participant, then it wins the race of Coordinator election, and declare itself as a Coordinator and start working normally as a Coordinator NameNode.

This is the process of election for new Coordinator. In case if all the Participant NameNodes are fully updated with the last Coordinator that has become failed, then they will have same priority numbers. In this case that Participant NameNode will be elected as Coordinator which have smallest serial number among them.

This approach of election algorithm has a disadvantage. since both the participant nodes are not time synchronized each other,it will affect the election algorithm. suppose the Participant Node which has the highest priority number may not be the one with least recent updation and if we choose that Prticipant Node as the next Coordinator Node, then we will miss the latest updation and it will be a wrong decision.So during the execution of election algorithm, we must also consider the time factor. For that some time synchronization mechanism is needed between the two Participant Nodes. Berkeley algorithm is used here for time synchronisation and is explained below.

2.4 Berkeley algorithm for time synchronization

The Berkeley algorithm, developed by Gusella and Zatti in 1989, does not assume that any machine has an accurate time source with which to synchronize. Instead, it opts for obtaining an average time from the participating computers and synchronizing all machines to that average. The machines involved in the synchronization each run a time demon process that is responsible for implementing the protocol. One of these machines is elected (or designated) to be the master. The others are slaves. Here the coordinator node acts as master and participant nodes acts as slaves for this algorithm. The server polls each machine periodically, asking it for the time. The time at each machine may be estimated by using Cristian’s method to account for network delays. Cristian’s Algorithm works between a coordinator node C and a participant node P is explained below:

- C requests the time from P
- After receiving the request from C, P prepares a response and appends the time T from its own clock.
- P then sets its time to be T + RTT/2

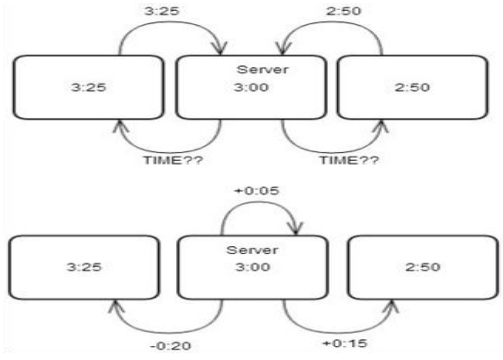

Fig. 4. Berkeley synchronization algorithm

When all the results are in, the master computes the average time (including its own time in the calculation). The hope is that the average cancels out the individual clock’s tendencies to run fast or slow. The

master then sends each slave the amount (positive or negative) that it should adjust its clock. Instead of sending the updated time back to the slaves, which would introduce further uncertainty due to network delays, it sends each machine the offset by which its clock needs adjustment. The operation of this algorithm is illustrated in Figure 4. Three machines have times of 3:00, 3:25, and 2:50. The machine with the time of 3:00 is the server (master). It sends out a synchronization query to the other machines in the group. Each of these machines sends a timestamps as a response to the query. The server now averages the three timestamps: the two it received and its own, computing (3:00+3:25+2:50)/3 = 3:05. Now it sends an offset to each machine so that the machine's time will be synchronized to the average once the offset is applied. The machine with a time of 3:25 gets sent an offset of -0:20 and the machine with a time of 2:50 gets an offset of +0:15. The server has to adjust its own time by +0:05.

III. Conclusion

The paper next explains the availability problems related to HDFS, the important of NameNode and the single point of failure (SPOF) problem of NameNode. To avoid the problem of single point of failure (SPOF) of NameNode, the paper proposes a highly available NameNode architecture. The proposed architecture uses the NameNode replication strategy where two replica of NameNode is used. Any updation in the coordinator NameNode will also be updated in the participant NameNode by using Two Phase Commit protocol. If the coordinator NameNode fails, one of the participants NameNode is elected as the coordinator NameNode by using an election algorithm called bully algorithm. For the proper selection of coordinator node using election algorithm, the participant nodes need to be synchronized. For time synchronization between the two participant nodes, Berkeley synchronization algorithm with Christian's method is used

In future, we will deploy this architecture on Hadoop and ensures its applicability. There is also another SPOF in the programming model of Hadoop that is Job Tracker which is the only entity for tracking MapReduce tasks. In future, we will extend our proposed model to address the SPOF problem of Job Tracker.

References

[1] Mohammad Asif Khan, Zulfiqar A. Memon,Sajid Khan, "Highly Available Hadoop NameNode Architecture", Department of CSE and ECE, 978-0-7695-4959-0/13 2013 IEEE
[2] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, "The Hadoop Distributed File System", Yahoo! Sunnyvale, California USA,978-1-4244-7153- 9/10/2010 IEEE
[3] Dhruba Borthakur, "The Hadoop Distributed File System: Architecture and Design", The Apache Software Foundation,2007
[4] Aaron Myers, "http://blog.cloudera.com/blog/2012/03/highavailability for-the- hadoop-distributed-file-system-hdfs/", cloudera,March 07, 2012.
[5] Tom White,"Hadoop: The Definitive Guide", 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2009.
[6] The Apache Software Foundation,"http:// http://hadoop.apache.org", Last release of march, 2013.
[7] Hadoop Wiki, "NameNode Failover", on Wiki Apache Hadoop, http://wiki.apache.org/hadoop/NameNodeFailover, July, 2013.
[8] Andrew S.Tanenbaum and Maarten Van Steen, "Distributed Systems: Principle and Paradigms", Pearson Prentice Hall, Upper Saddle River, NJ 07458.
[9] Paul Krzyzanowski, "Lectures on distributed systems Clock Synchronization", Rutgers University CS 417: Distributed Systems 2000-2009 Paul Krzyzanowski