SUSTAINABILITY OF HADOOP CLUSTERS

Luis Bautista, Alain April
ETS, University of Quebec, Software Engineering Department, 1100 Notre-Dame, Montreal, Canada
lebautis@correo.uaa.mx, alain.april@etsmtl.ca

Keywords: Cloud Computing, High Availability Cluster, Hadoop Distributed File System, ZooKeeper

Abstract: Hadoop is a set of utilities and frameworks for the development and storage of distributed applications in cloud computing, the core component of which is the Hadoop Distributed File System (HDFS). NameNode is a key element of its architecture, and also its "single point of failure". To address this issue, we propose a replication mechanism that will protect the NameNode data in case of failure. The proposed solution involves two distinct components: the creation of a BackupNode cluster that will use a leader election function to replace the NameNode, and a mechanism to replicate and synchronize the file system namespace that is used as a recovery point.

1 INTRODUCTION

Cloud computing is a new technology aimed at processing and storing very large amounts of data. It is an Internet-based technology, in which several distributed computers work together to process information in a more efficient way and deliver results more quickly to the users who require them. In general, cloud computing users don't own the physical infrastructure. Instead, they rent usage of infrastructure, platform or software from a third-party provider. The delivery of computer infrastructure, platform of software or applications, typically is known as Cloud Services (Jin, Ibrahim et al. 2010).

There are several Cloud Service Providers (CSP) for the different type of services, for example Amazon EC2, SalesForce.com, 3tera Inc, and Eucalyptus, among many others, making use of distributed computing technologies. One of these technologies is called the distributed file system (DFS), which allows access to files from multiple computers accessible via the Internet. The Google File System (GFS) and the Hadoop Distributed File System (HDFS) are two examples of DFS implementations.

DFS, like the open source project HDFS, are designed to store very large files, across multiple computers, where exceptional reliability is provided by its replication mechanisms. Replication across multiple computers can replace the need for RAID (redundant array of independent disk) storage technology. Also, HDFS is designed to run on a large number of commodity computers concurrently. Commodity computers are computer systems manufactured by multiple vendors, incorporating components based on open standards. A governing principle of commodity computing is that it is better to have more lower performance and lower cost hardware working in parallel than it is to have fewer, but more expensive computers. The key to using commodity computers in large numbers is a replication mechanism that provides high fault tolerance on low-cost hardware.

However, DFS must provide guaranteed high availability. HDFS has been designed with a master/slave architecture of clusters, which consists

of a single NameNode (NN), the master server that manages the file system namespace and regulates access to files by clients. In addition, a number of DataNodes (DN) manage large amounts of storage (Borthakur, 2008). The existence of a unique NameNode in an HDFS greatly simplifies the architecture of this technology, however it is also its weakness. We call this weakness a *single point of failure* (SPoF). When an HDFS NameNode fails, fixing it currently requires a manual recovery.

2 SINGLE POINT OF FAILURE IN HDFS

We have stated that an HDFS cluster has two types of nodes (computers): a master node called a *NameNode (NN)* and a number of slaves nodes called *DataNodes (DN)*. The NN manages the file system namespace, which is where we maintain the file system tree and the metadata for all the files and directories. This information is persistently stored on the local disks in two files: the *namespace image* and the *edit log*. The NN also keeps track of the DN on which all the blocks for a given file are located. However, it does not store block locations persistently, since this information is reconstructed from NNs when the system starts (White, 2009).

The single point of failure (SPoF) in an HDFS cluster is the NN, while the loss of any other node (intermittently or permanently) does not result in a data loss. So, NN loss results in HDFS cluster unavailability. The permanent loss of NN data would render the HDFS cluster inoperable (Yahoo, 2010). For this reason, it is important to make the NN resilient to failure, and HDFS provides a manual mechanism for achieving this.

The steps of the mechanism are as follows. First, a backup up is made of the files that make up the persistent state of the file system metadata, where the usual configuration choice is to write to the local disk as well as to a remote NFS mount. Then, a *secondary NameNode (SNN)* must be run, which will periodically merge the namespace image with the edit log. This is necessary to prevent the edit log from becoming too large. In HDFS, it is recommended that the SNN run on a separate physical computer, since this merge requires as much CPU and memory as the NN (Apache, 2010). However, when a failure occurs, a manual intervention is necessary to copy the NN metadata files, which are on the NFS, to the SNN that will become the new NN.

There are currently some efforts planned to convert the SNN to a standby node, which, besides handling merging, could also maintain the up-to-date state of the namespace, by processing constant edits from the NN, and of the checkpoint node (which creates the checkpoints of the namespace). This standby node approach has been named a *Backup Node (BN)* (Apache, 2008).

To resolve the SPoF, the BN would provide real-time streaming of edits from an NN to a BN. This would allow constant updating of the namespace state. The BN would also conduct a checkpointing function, ensuring the availability of the HDFS namespace in memory and getting rid of the current need to store the namespace on disk. Finally, the BN proposal would offer the availability of a standby node. This node, coupled with an automatic switching (failover) function, would eliminate potential data loss, unavailability, and manual interventions into HDFS NN failures.

However, if the BN fails, what will take its place?

3 PROPOSED SOLUTION

In this paper, we propose a distributed solution to the problem of NN and BN failures, which makes use of a coordination scheme and leader election function within BN replicas. This can be achieved using a service, such as ZooKeeper, for maintaining configuration information, for naming, and for distributed synchronization and group coordination.

3.1 Distributed Applications with ZooKeeper

ZooKeeper is a service that allows distributed processes to coordinate with each other through a shared hierarchal namespace of data registers. It has proven that it can be useful for large distributed systems applications (Apache, 2008).

One of the main failure recovery problems with distributed applications is partial failure. For example, when a message is sent across the network and it fails, the message will not be received, or when the receiver's process dies, the sender does not

know the reason for the failure. ZooKeeper provides a set of tools to protect distributed applications when this type of failure occurs.

Also of interest is that Zookeeper runs on a cluster of computers called an *ensemble*, and is designed to be highly available due to its *replicated mode*. It has great potential to help solve the SPoF problem of HDFS. We propose to use it to design and manage a high availability BN cluster. With this approach, if the *Primary Backup Node* (PBN) fails, then an election mechanism for choosing a new PBN is initiated. There could be a number of *Replicated Backup Nodes* (RBN), as shown in Figure 1.

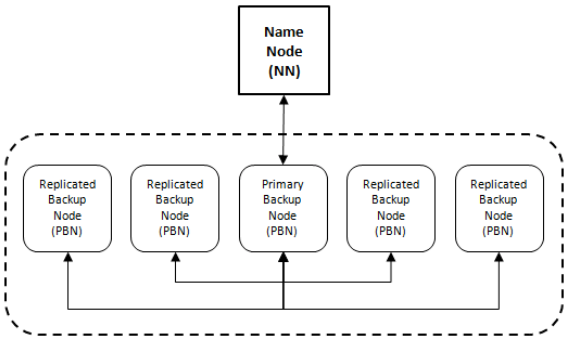

Figure 1: ZooKeeper Service for PBN Election.

3.2 Primary Backup Node Election

How would the PBN election mechanism work? It would be based on the use of a protocol called *Zab* (Red Junqueira, 2008). This protocol is already implanted in ZooKeeper and runs in two phases, (which may be repeated indefinitely):

Phase 1: *Leader Election.* The computers in an *ensemble* (group of RBNs) go through a process of electing a distinguished member, called a *leader* (PBN). The other machines are termed *followers*. This phase is finished once a majority (or *quorum*) of followers has synchronized their state with that of the leader.

Phase 2: *Atomic Broadcast*. All write requests are forwarded to the leader (PBN), which broadcasts the update to the followers (RBNs). When a majority has processed the change, the leader commits to the update, and the client receives a response to the effect that the update has succeeded. The protocol for achieving consensus is designed to be atomic, so a change either succeeds or fails. It resembles a two-phase commit.

Thus, if the NN fails, the PBN will take its place and begin a leader election process within the group of RBNs. This will result in the selection of a new PBN that will take the place of the old PBN.

There are two important issues related to this HDFS high availability proposal: replication and synchronization of data. Replication is at the core of our proposal, and would use an efficient and flexible synchronization mechanism that must support different workloads and offer optimal performance. In our proposal, the file system namespace, which must be replicated and which is included in the *fsimage* file, is merged with the edit log to obtain a persistent checkpoint of the file system.

Therefore, we must design a mechanism that would create a reliable replication service for those files. At the same time, this mechanism must provide a recovery service during failures. There are different ways to achieve this goal. We have investigated ZooKeeper and BooKeeper as potential solutions.

3.3 Log Stream of Records with BooKeeper

The initial motivation for investigating BooKeeper was that the NN of HDFS uses logs (*edit logs*) for recovery in case of failure. BooKeeper was designed as a replication service to reliably log streams of records, where a BooKeeper Client (BC) receives *ledgers*, which are entries of log streams from a *client application*, and stores them to sets of BooKeeper servers called *bookies* (Apache, 2010). Besides providing high-availability services, BooKeeper provides good performance by using striping and scalability during quorums.

In our proposed solution, we intend to create ledgers, which will contain the *namespace image* of HDFS, and write them into bookies. This process could be performed by the BooKeeper Client (BK), which would run concurrently with a BooKeeper Application (BA), as demonstrated in Figure 2. The

bookies would store the content of the ledgers in an *ensemble* of bookies, storing into it the content of different ledgers. To ensure good performance, BooKeeper would store each bookie of an *ensemble* as a fragment of a ledger. That is, each entry would be written to sub-groups of bookies of the *ensemble*.

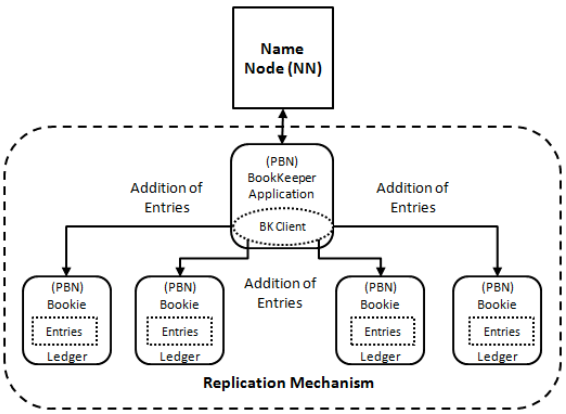

Figure 2: Log Stream of Records with BooKeeper.

With this proposal, the BN would maintain an in-memory data structure. This data structure would add entries from the NN, and, at the same time, the BN can process one *asyncAddEntry* to queue up changes. This would ensure a high change throughput. Thus, our proposed solution to the HDFS SPoF has the potential to create a reliable mechanism of replicas and synchronization.

4 FUTURE WORK

In the next step of this research, we will conduct a detailed design, based on the solution proposal described here, of an amended HDFS. This will allow us to assess the many possibilities of embedding this solution within the distributed file system (HFDS). Potential detailed solutions and tradeoffs, will be investigated, and suitable families of protocols to resolve this problem will be identified. Once the selected design has been implemented, we will carry out a case study assessing the feasibility of using a highly available and reliable coordination system imbedded in HDFS to address the SPoF problem.

5 CONCLUSION

This paper has presented a potential solution to the problem of the "single point of failure" of HDFS aimed at implementing a mechanism in HDFS similar to BooKeeper. The proposed solution has two distinct components: 1) a BackupNode cluster which uses a leader election function and which can replace the NameNode in case of failure; and 2) a mechanism to replicate and synchronize the file system namespace that is used as a recovery point.

REFERENCES

Apache Hadoop, 2010. http://hadoop.apache.org/

Apache Software Foundation, 2008. Streaming Edits to a Backup Node, https://issues.apache.org/jira/browse/HADOOP-4539 .

Apache Software Foundation, 2008. ZooKeeper Overview http://hadoop.apache.org/zookeeper/docs/current/zookeeperOver.html

Apache Software Foundation, 2010. BooKeeper Overview. http://hadoop.apache.org/zookeeper/docs/r3.3.0/bookkeeperOverview.html

Carolan, G., 2009. Introduction to Cloud Computing Architecture. Sun Microsystems.

Dhruba, B., 2008. Hadooop Distributed File System Architecture.

Jin, H., Ibrahim, S., Bell, T., Qi,L., Cao, H., Wu, S., and Shi, X. (2010) *Tools and Technologies for Building Clouds*, Cloud Computing: Principles, Systems and Applications, Computer Communications and Networks, Springer-Verlag.

Red, B., Junqueira, F.P., 2008. A Simple Totally Ordered Broadcast Protocol. *In proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware* (LADIS), Yorktown Heights, New York, September 15 - 17, vol. 341:2008).

White, T., 2009. Hadoop: The Definitive Guide, O´Reilly Media, Inc.

Yahoo! Inc, 2010. Managing a Hadoop Cluster, http://developer.yahoo.com/hadoop/tutorial/module7.html#configs .