American Journal of Engineering Research (AJER)
e-ISSN : 2320-0847 p-ISSN : 2320-0936
Volume-4, Issue-3, pp-52-54
www.ajer.org

Research Paper Open Access

Recuperation of data node: An alternative way to ensure the data recovery in Hadoop architecture.

[1,] P.Sai Kiran
(Computer Science and Engineering, Vidya Jyothi Institute of Technology (JNTU-H), AP, India)

ABSTRACT: *Hadoop is a software framework that supports data intensive distributed application. Hadoop creates clusters of machine and coordinates the work among them. It include two major component, HDFS (Hadoop Distributed File System) and MapReduce. HDFS is designed to store large amount of data reliably and provide high availability of data to user application running at client. It creates multiple data blocks and store each of the block redundantly across the pool of servers to enable reliable, extreme rapid computation. MapReduce is software framework for the analyzing and transforming a very large data set in to desired output. This paper focus on how the data can be recovered by a variation in the architecture ensuring the continuation of computing if the DataNode fails.*

KEYWORDS: *Hadoop, HDFS, clustering*

I. INTRODUCTION

Hadoop distribute and parallelize data processing across many nodes in a compute cluster, speeding up large computations and hiding I/O latency through increased concurrency. It is well suited for large data processing like searching and indexing in huge data set. Hadoop includes Hadoop Distributed File System (HDFS) and MapReduce. It is not possible for storing large amount of data on a single node, therefore Hadoop use a new file system called HDFS which split data into many smaller parts and distribute each part redundantly across multiple nodes. MapReduce is a software framework for the analysis and transformation of very large data sets. Hadoop uses MapReduce function for distributed computation. MapReduce programs are inherently parallel. Hadoop take advantage of data distribution by pushing the work involved in analysis to many different servers. Each server runs the analysis on its own block from the file. Results are combined in to single result after analyzing each piece. MapReduce framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks [1].

This paper gives an overview of an alternative way to recuperate the DataNode loss in the hadoop architecture.

II. HADOOP DISTRIBUTED FILE SYSTEM (HDFS)

HDFS is a distributed file system designed to run on commodity hardware. HDFS is highly fault tolerant and is designed to be deployed on low cost hardware. HDFS is suitable for applications that have large dataset.

HDFS maintain the metadata in a dedicated server called NameNode and the application data are kept in separated nodes called DataNode. These server nodes are fully connected and they communicate using TCP based. protocols. In HDFS the file content are replicated on multiple DataNodes for reliability.

2.1 NameNodes

HDFS name space is a hierarchy of files and directories. Files and directories are represented on the NameNode using inodes which record attributes like permissions modification and access time, namespace and disk space quotas. The file content is split into blocks (typically 128MB) each block of file is independently replicated at multiple DataNodes. NameNode maintains the mapping of file blocks to DataNodes. An HDFS client waiting to read a file first contact the NameNode for the locations of data blocks comprising the file and

then reads block content from the DataNode closest to the client. When writing the data, the client requests the NameNodes to nominate a set of DataNodes to host the block replicas.

2.2 DataNodes

Each data block is represented by two files in the host native file system, one file contains the data itself and the other contains block‟s metadata. Handshake is performed between all DataNodes and the NameNode at startup. During handshake, the namespace ID and software version of DataNode is verified with the NameNode. If it does not match with that of NameNode, then that DataNode will automatically shuts down. Namespace ID is assigned to the file system instance when it is formatted. A newly initialized DataNode without any namespace ID can join the cluster and the will receive the cluster‟s namespace ID. Each DataNode persistently store its unique storage ID, which help to recognize it after restarting it with a different IP address or port. Each DataNode send block report to the NameNode to identify the block replicas in its possession. First block report is send during DataNode registration and the subsequent block reports are sent at every hour. This helps the NameNode to keep an up-to-date view of where block replicas are located on the cluster.

Each DataNode send heartbeat to NameNode to confirm that it is operating and its block replicas are available. Default heartbeat interval is 3 seconds and if no heartbeat signal is received at NameNode in 10 minutes, the NameNode will mark the DataNode as unavailable. NameNode schedules creation of new replica of those blocks on another DataNode. NameNode use replies to the heartbeat to send instruction to DataNodes [1].

2.3 CopyNode

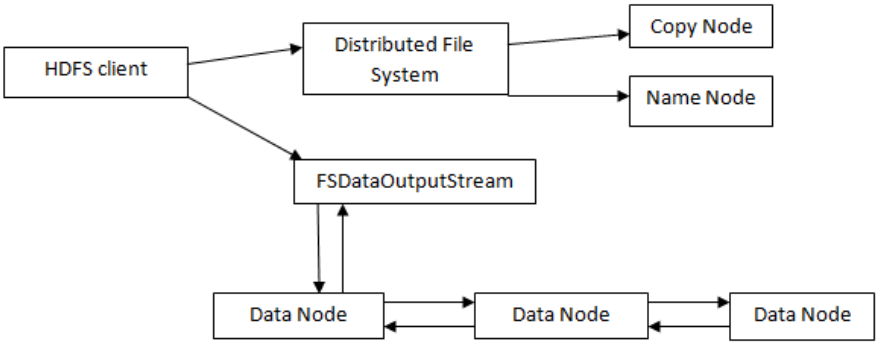
Copy node is a space in the hadoop architecture where the data of the current data node is stored. During handshake, the namespace ID and software version of DataNode is verified with the NameNode. Namespace ID is assigned to the file system instance when it is formatted. Copy node is primarily a defensive mechanism when the write operation fails on the data node for any reason.

III. ANATOMY OF FILE WRITE

Client creates the file on DistributedFileSystem (step 1). DistributedFileSystem makes an RPC call to the copynode to create a new file in the filesystem's namespace, with no blocks associated with it (step 2). DistributedFileSystem makes an RPC call to the namenode to create a new file in the filesystem's namespace, with no blocks associated with it (step 3).

The namenode performs various checks to make sure the file doesn't already exist, and that the client has the right permissions to create the file. If these checks pass, the namenode makes a record of the new file; otherwise, file creation fails and the client is thrown an IOException. The DistributedFileSystem returns an FSDataOutputStream for the client to start writing data to. Just as in the read case, FSDataOutputStream wraps a DFSOutputStream, which handles communication with the datanodes and namenode.As the client writes data (step 4), DFSOutputStream splits it into packets, which it writes
to an internal queue, called the *data queue*.

The data queue is consumed by the Data Streamer, whose responsibility it is to ask the namenode to allocate new blocks by picking a list of suitable datanodes to store the replicas. The list of datanodes forms a pipeline—we'll assume the replication level is three, so there are three nodes in the pipeline. The DataStreamer streams the packets to the first datanode in the pipeline, which stores the packet and forwards it to the second datanode in the pipeline. Similarly, the second datanode stores the packet and forwards it to the third (and last) datanode in the pipeline (step 5). DFSOutputStream also maintains an internal queue of packets that are waiting to be acknowledged by datanodes, called the *ack queue*. A packet is removed from the ack queue only when it has been acknowledged by all the datanodes in the pipeline (step 6)



If a datanode fails while data is being written to it, then the following actions are taken, which are transparent to the client writing the data. First the pipeline is closed, and any packets in the ack queue are added to the front of the data queue. Since the current data is already present in the copy node, the data is copied into all the three data nodes. As long as dfs.replication.min replicas (default one) are written, the write will succeed, and the block will be asynchronously replicated across the cluster until its target replication. Factor is reached (dfs.replication, which defaults to three). When the client has finished writing data, it calls close () on the stream (step 7). This action flushes all the remaining packets to the datanode pipeline and waits for acknowledgments before contacting the namenode to signal that the file is complete (step8). The namenode already knows which blocks the file is made up of (via DataStreamer asking for block allocations), so it only has to wait for blocks to be minimally replicated before returning successfully [2].

IV. CONCLUSION

Though it is unlikely, that multiple datanodes fail while a block is being written, it should be a defensive practice to not leave that to sheer chance. The change in the hadoop architecture ensures continual processing without a sudden breakdown.

REFERENCES

Journal papers

[1] Shvachko K, Kuang H, Radia S and Chansler R. The Hadoop Distributed File System in Proceedings of the 26th IEEE Symposium on Massive Storage Systems and Technologies, 2010.

Books

[2] Tom White , Hadoop the definitive guide.