Cloud Computing | Ankit Akash Jha | 2016CS10342

HDFS

HDFS is a distributed, highly tolerant **low-cost-hardware-deployment** targeted file system. It's key emphasis is on providing **high throughput** access with lower regards to latency rate. It is defined more for batch processing compared to user based use. It's based on write once read many access model. It moves computation rather than data and is designed for to easy **platform portability**.

An HDFS cluster consists of Master Slave policy. Single **Name Node** acts as master with multiple **Data Nodes** as slave. Name nodes regulates *client access* and maintains *file system namespace* and executes their *operations* and determines *block- data node mappings*. Data nodes **manages storage** attached to their respective nodes and **serves read write requests** for the clients. They also perform **block creation, deletion** etc. HDFS utilises the fact that network bandwidth between machines in the same rack is greater than network bandwidth between machines in different racks by **evenly distributing replicas** in the cluster which makes it easy to balance load on component failure.

Files are divided into chunks of identical size(~) called blocks. These **blocks are replicated** for fault tolerance. The number of copies of blocks is called replication factor and they are stored on Name Nodes. Name node uses a file transaction log, called **Edit log** to continuously record every change in file system metadata. Edit log is saved in a file in local host os system, HDFS also maintains a file **Fsimage** which stores the entire *file system namespace*, including *mapping* of blocks to files and file system *properties*. Name node on starting performs **checkpoint**. Under this it read Fsimage and edit log, applies all the transaction from edit log to the in memory representation of FSimage and flushes out new version on newer FSimage on the disk. It then discards the old edit log as its transactions have been applied to the FSimage. HDFS client software implements **checksum** on the contents. Plus the Name node keeps multiple copies of Fsimage and edit log as well. Thus Name node is the single point of failure of the system. If it fails, manual intervention is needed.

GFS

It is again a **scalable distributed** file system for **large** distributed **data intensive** applications. The system is built using many inexpensive components which often fail. The target is to have **good fault tolerance** and **auto recovery** with *high performance on a large number of clients.*

A GFS Cluster typically consists of a single **master** and multiple **chunk servers**. Files are divided into multiple fixed size **chunks**. These chunks are identified by an *immutable globally unique* 64 bit chunk handler assigned to them at time of allocation by the master. Chunks are *replicated* on *multiple chunk server* for **reliability**. Master maintains all file system metadata from *namespace* to *mapping between chunks and files*, *access control information* and the *current locations of chunks*. It doesn't keep persistent record of which chunk servers have replica of a given chunk. It finds them by simply polling at the startup.

The key differences between GFS and HDFS are :

1. The structure in GFS is divided into *master and chunk server* while in HDFS they are called *Name node and Data node.*
2. In GFS TCP connections with *pipelining* for data transfer are used for communication while in HDFS *RPC based protocol* on top of TCP/IP are used.
3. GFS is based on *Append once Read many mode*l providing multiple clients to append to the same file concurrently. HDFS is *Write once Read many model*.
4. GFS supports *snapshot* while HDFS doesn't. Snapshot creates a copy of a file or a directory tree at low cost. HDFS is supposed to provided it in newer versions.
5. GFS doesn't save the *location of replica chunks* as metadata while HDFS does.
6. GFS is *proprietary*, exclusive for google use only while HDFS is *open source*.

The model have a lot of **similarities** starting from the basic **design goals** they address, both targeting large file support, with fault tolerance and batch processing. Both are designed to be **scalable**. Both have similar file management hierarchy. Both have **fixed size chunk** divisions. Uses **replication strategy** by spreading across different racks for fault recovery.