## The communication Protocols

All HDFS communication protocols are layered on top of the TCP/IP protocol. A client establishes a connection to a configurable TCP port on the NameNode machine. It talks the ClientProtocol with the NameNode. The DataNodes talks to the NameNode using the DataNode Protocol. A ***Remote Procedure Call (RPC)*** abstraction wraps both the Client Protocol and The DataNode Protocol. By design, the NameNode never initiates any RPCs. Instead, it only responds to RPC requests issued by DataNodes or clients.

## Robustness

The primary objective of HDFS is to store data reliably even in the presence of failures. The three common types of failures are NameNode failures, DataNode failures and network partitions.

1. **Data disk failure, Heartbeats and Re-replication** Each DataNode sends a Heartbeat message to the NameNode periodically. A network partition (caused due to the failure of network devices) can cause a subset of DataNodes to lose connectivity with the NameNode. The NameNode detects this condition by the absence of a Heartbeat message. The NameNode marks DataNodes without recent Heartbeats as dead and does not forward any new IO requests to them. Any data that was stroed in a dead DataNode is not available to HDFS anymore. DataNode death may cause the replication factor of some blocks to fall below their specified value. The NameNode constantly tracks which blocks need to be replicated and initiates replication whenever necessary. The necessity for re-replication may rise due to many reasons: a DataNode may become unavailable, a replica may become corrupted, a hard disk on a DataNode may fail, or the replication factor of a file may be increased. The time-out to mark DataNodes dead is conservatively long (over 10 minutes by default) in order to avoid replication storm caused by state flapping of DataNodes. Users can set shorter interval to mark DataNodes as stale and avoid stale nodes on reading and/or writing by configuration for performance sensitive workloads.

2. **Cluster Rebalancing** The HDFS architecture is compatible with data rebalancing schemes. A scheme might automatically move data from one DataNode to another if the free space on a DataNode falls below a certain threshold. In the event of a sudden high demand for a particular file, a scheme might dynamically create additional replicas and rebalance other data in the cluster. These types of data rebalancing schemes are not yet implemented.

3. **Data integrity** It is possible that a block of data fetched from a DataNode arrives corrupted. This corruption can occur because of faults in a storage device, network faults, or buggy software. The HDFS client software implements checksum checking on the contents of HDFS files. When a client creates an HDFS files, it computes a checksum of each block of the file and stores these checksums in a separate file in the same HDFS namespace. When a client retrieves file contents it verifies that the data it received from each DataNode matches the checksum sorted in the associated checksum file. If not, then the client can opt to retrieve that block from another DataNode that has a replica of that block.

4. **Metadata Disk Failure** The FsImage and the EditLog are central data structures of HDFS. A corruption of these files can cause the HDFS instance to be non-functional. For this reason, the NameNode can be configured to support maintaining multiple copies of the FsImage and EditLog. Any update to either the FsImage or EditLog causes each of these FsImages and EditLogs to get updated synchronously. This synchronous updating of multiple copies of the FsImage and EditLog may degrade the rate of namespace transaction per second that a NameNode can support. However, this degradation is acceptable because even though HDFS applications are very data intensive in nature, they are not metadata intensive. When a NameNode restarts it selects the latest consistent FsImage and EditLog to use. Another option to increase resilience against failures is to enable High Availability using multiple NameNodes either with a shared storage on NFA or using a distributed EditLog (called Journal). The latter is the recommended approach.

5. **Snapshots** Snapshots support storing a copy of data at a particular instant of time. One usage of the snapshot feature may be to roll back a corrupted HDFS instance to a previously known good point in time.

## Data Organisation

1. **Data blocks** HDFS is designed to support very large files. Applications that are compatible with HDFS are those that deal with large data sets. These applications write their data only once but they read it one or more time and require these reads to be satisfied at streaming speeds. HDFS supports write-once-read-many semantics on files. A typical block size used by HDFS is 128MB. Thus, an HDFS file is chopped up into 128 MB chunks, and if possible, each chunk will reside on a different DataNode.

2. **Replication Pipelining** When a client is writing data to an HDFS file with a replication factor of three, the NameNode retrieves a list of DataNodes using a replication target choosing algorithm. This list contains the DataNodes that will host a replica of that block. The client then writes to the first DataNode. The first DataNode start receiving the data in portions writes each portion to its local repository and transfers that

portion to the second DataNode in the list. The second DataNode, in turn starts receiving each portion of the data block, writes that portion to its repository and then flushes that portion to the third DataNode. Finally, the third DataNode writes the data to its local repository. Thus, a DataNode can be receiving data from the previous one in the pipeline and at the same time forwarding data to the next one in the pipeline. Thus, the data is pipelined from one DataNode to the next.

## Sources:

1. Main
2. NameNode Failure - shared NFA storage (Metadata disk failure)
3. NameNode failure - Journaling (Metadata disk failure)