**HDFS Federation and High availability**

Before Hadoop 2 comes to the picture, Hadoop clusters were living with the fact that Name Node has placed limits on the degree to which they could scale. Some of the clusters were able to scale beyond 3,000 or 4,000 nodes. NameNode’s require to maintain records for each block of data stored in the cluster turned out to be the most significant factor limiting the greater cluster growth. When we have too many blocks, it becomes increasingly difficult for the NameNode to scale up as the Hadoop cluster scales out.

The solution to expanding Hadoop clusters indefinitely is to federate the NameNode. Specifically, we must set it up so that we have multiple NameNode instances running on their own, dedicated master nodes and then making every NameNode responsible only for the data file blocks in its own name space.

Often in Hadoop’s infancy, a great amount of discussion was focused on the NameNode’s representation of a single point of failure (SPOF's). Hadoop, entirely, has always had a robust and failure-tolerant architecture design, with the exception only in this key area. As we already know, without the NameNode, there’s no Hadoop cluster

Using Hadoop 2,  now we can configure HDFS so that there exists two kind of NameNodes:

1.    An **Active NameNode** and,

2.    A **Standby NameNode**

The Standby NameNode required being on a dedicated master node that’s configured very similar to the master node used by the Active NameNode.

The Standby NameNode’s work is not be sitting idly while the NameNode manages all the block address requests. The Standby NameNode, charged with the job of keeping the state of the block locations and block metadata in memory, managing the HDFS check-pointing duties. The Active NameNode now writes journal entries on file changes to the majority of the ***JournalNode*** services, which executes on the master nodes. (If a failure occurs, the Standby Node first reads all of the completed journal entries (where a majority of Journal Nodes have an entry, in simple words), to make sure that the new Active NameNode is perfectly consistent with the state of the cluster.

**Hdfs handles failure while writing data:**

hdfs handles failure by replicating the data while writing into it.

HDFS replication enables you to copy (replicate) your HDFS data from one HDFS service to another, synchronizing the data set on the destination service with the data set on the source service, based on a specified replication schedule

An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later. Files in HDFS are write-once and have strictly one writer at any time.  
The NameNode makes all decisions regarding replication of blocks. It periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. Receipt of a Heartbeat implies that the DataNode is functioning properly. A Blockreport contains a list of all blocks on a DataNode.  
  
 HDFS’s placement policy is to put one replica on one node in the local rack, another on a node in a different (remote) rack, and the last on a different node in the same remote rack. This policy cuts the inter-rack write traffic which generally improves write performance.  
  
The default block placement policy is as follows:

* Place the first replica somewhere – either a random node (if the HDFS client is outside the Hadoop/DataNode cluster) or on the local node (if the HDFS client is running on a node inside the cluster).
* Place the second replica in a different rack.
* Place the third replica in the same rack as the second replica
* If there are more replicas – spread them across the rest of the racks.