**HDFS**

**Hadoop HDFS – Introduction**

**Hadoop Distributed Filesystem – HDFS** is the world’s most reliable storage system. HDFS is a Filesystem of Hadoop designed for storing very large files running on a cluster of commodity hardware. It is designed on principle of storage of less number of large files rather than the huge number of small files. It provides fault tolerant storage layer for Hadoop and its other components. Replication of data helps us to attain this feature. It stores data reliably even in the case of hardware failure. It provides high throughput access to application data by providing the data access in parallel.

**HDFS Nodes**

As we know Hadoop works in**master-slave** fashion, HDFS also has 2 types of nodes that work in the same manner. There are **namenode(s)** and **datanodes** in the cluster.

### HDFS Master (Namenode)

Namenode regulates file access to the clients. It maintains and manages the slave nodes and assign tasks to them. Namenode executes file system namespace operations like opening, closing, and renaming files and directories. It should be deployed on reliable hardware.

### HDFS Slave (Datanode)

There are a number of slaves or datanodes in HDFS which manage storage of data. These slave nodes are the actual worker nodes which do the tasks and serve read and write requests from the file system’s clients. They also perform block creation, deletion, and replication upon instruction from the NameNode. Once a block is written on a datanode, it replicates it to other datanode and process continues until the number of replicas mentioned is created. Datanodes can be deployed on commodity Hardware and we need not deploy them on very reliable hardware.

## HDFS Daemons

There are 2 daemons which run for HDFS for storage of data.

* **Namenode:** This is the daemon that runs on all the masters. Name node stores metadata like filename, the number of blocks, number of replicas, a location of blocks, block IDs etc. This metadata is available in memory in the master for faster retrieval of data. In the local disk, a copy of metadata is available for persistence. So name node memory should be high as per the requirement.
* **Datanode:** This is the daemon that runs on the slave. These are actual worker nodes that store the data.

## Data storage in HDFS

Whenever any file has to be written in HDFS, it is broken into small pieces of data known as blocks. HDFS has a default block size of **128 MB** which can be increased as per the requirements. These blocks are stored in the cluster in distributed manner on different nodes. This provides a mechanism for [**MapReduce**](http://data-flair.training/blogs/hadoop-mapreduce-introduction-tutorial-comprehensive-guide/)to process the data in parallel in the cluster.

Multiple copies of each block are stored across the cluster on different nodes. This is a replication of data. By default, HDFS has a replication factor of 3. It provides fault tolerance, reliability, and [**high availability**](https://data-flair.training/blogs/hadoop-high-availability-tutorial/).

A Large file is split into n number of small blocks. These blocks are stored at different nodes in the cluster in a distributed manner. Each block is replicated and stored across different nodes in the cluster.

## Rack Awareness in Hadoop HDFS

Hadoop runs on a cluster of computers which are commonly spread across many racks. NameNode places replicas of a block on multiple racks for improved fault tolerance. NameNode tries to place at least one replica of a block in each rack, so that if a complete rack goes down then also system will be highly available Optimizing replica placement distinguishes HDFS from most other distributed file systems. The purpose of a rack-aware replica placement policy is to improve data reliability, availability, and network bandwidth utilization.

## HDFS Architecture

This architecture gives you a complete picture of HDFS. There is a single namenode which stores metadata and there are multiple datanodes which do actual storage work. Nodes are arranged in racks and Replicas of data blocks are stored on different racks in the cluster to provide fault tolerance. In remaining section of this tutorial we will see, how read and write operations are performed in HDFS? To read or write a file in HDFS, the client needs to interact with Namenode. HDFS applications need a write-once-read-many access model for files. A file once created and written cannot be edited.

### Blocks

As HDFS splits huge files into small chunks known as blocks. Block is the smallest unit of data in a filesystem. We (client and admin) do not have any control on the block like block location. Namenode decides all such things.

HDFS default block size is 128 MB which can be increased as per the requirement. This is unlike OS filesystem where the block size is 4 KB.

If the data size is less than the block size, then block size will be equal to the data size. For example, if the file size is 129 MB, then 2 blocks will be created for it. One block will be of default size 128 MB and other will be 1 MB only and not 128 MB as it will waste the space (here block size is equal to data size). Hadoop is intelligent enough not to waste rest of 127 MB. So it is allocating 1 MB block only for 1 MB data.

The major advantage of storing data in such block size is that it saves disk seek time and another advantage is in the case of processing as[**mapper**](http://data-flair.training/blogs/mapper-in-hadoop-mapreduce/)processes 1 block at a time. So 1 mapper processes large data at a time.

The file is split into blocks and each block is stored at different nodes with default 3 replicas of each block. Each replica of a block is stored at the different node to provide fault tolerant feature and the placement of these blocks on the different node is decided by Name node. Name node makes it as much distributed as possible. While placing a block on a data node, it considers how much a particular data node is loaded at that time.

### Replication

Hadoop HDFS creates duplicate copies of each block. This is called replication. All blocks are replicated and stored at different nodes across the cluster. It tries to put at least 1 replica in every rack.

What do you mean by rack?

Datanodes are arranged in racks. All the nodes in a rack are connected by a single switch so if a switch or complete rack is down, data can be accessed from another rack. We will see it further in rack awareness section.

As seen earlier also, default replication factor is 3 and this can be changed to the required values according to the requirement by editing the configuration files (hdfs-site.xml)

### High Availability

Replication of data blocks in HDFS and storing at multiple nodes across cluster provides high availability of data. Even if a network link or node or some hardware goes down, we can easily get the data from the different path or different node as data is minimally replicated at 3 nodes. This is how high availability feature is supported by HDFS.

### Data Reliability

As we have seen in high availability that data is replicated in HDFS, It is stored reliably as well. Due to replication, blocks are highly available even if some node crashes or some hardware fails. We can balance the cluster (if 1 node goes down, block that was stored at that node become under replicated or if a node which has gone down suddenly becomes active, block at that node is over replicated. We need to balance the cluster to create or destroy the replica as per the situation) in such case by making the replication factor to desired value by just running few commands. This is how data is stored reliably and provides fault tolerant and high availability.

### Fault Tolerant

HDFS provides fault tolerant storage layer for Hadoop and other components in the ecosystem. HDFS works with commodity hardware (systems with average configurations) that has high chances of getting crashed any time. Thus, to make the entire system highly fault-tolerant, HDFS replicates and stores data in different places. Any data on HDFS gets stored at 3 different locations by default. So, even if one of them is corrupted and the other is unavailable for some time for any reason, then data can be accessed from the third one. Hence, there is no chance of losing the data. This replication factor helps us to attain the feature of Hadoop called Fault Tolerant.

### Scalability

Scalability means expanding or contracting the cluster. In HDFS, scalability is done in 2 ways.

a) We can add more disks on nodes of the cluster.

For doing this, we need to edit the configuration files and make corresponding entries of newly added disks. Here we need to provide down time though it is very less. So people generally prefer second way of scaling which is horizontal scaling.

b) Another option of scalability in HDFS is of adding more nodes to the cluster on the fly without any downtime. This is known as horizontal scaling.

We can add as many nodes as we want in the cluster on the fly in real time without any downtime. This is a unique feature provided by Hadoop.

### High throughput access to application data

HDFS provides high throughput access to application data. Throughput is the amount of work done in a unit time. It describes how fast the data is getting accessed from the system and it is usually used to measure the performance of the system. In HDFS, when we want to perform a task or an action, then the work is divided and shared among different systems. So all the systems will be executing the tasks assigned to them independently and in parallel. So the work will be completed in a very short period of time. In this way, the HDFS gives good throughput. By reading data in parallel, we decrease the actual time to read data tremendously.

## Hadoop HDFS Operations

In Hadoop, we need to interact with the file system either by programming or by command line interface (CLI). Learn how to interact with HDFS using CLI from [this commands manual](http://data-flair.training/blogs/interact-hadoop-hdfs-commands-perform-operations/).

HDFS is having lot many similarities with[**Linux**](http://data-flair.training/blogs/frequent-linux-commands-beginners-tutorial/)file system. So we can do almost all the operation we can do with a local file system like create a directory, copy the file, change permissions etc.

HDFS also provides different access rights like read, write and execute to users, groups, and others.

We can browse the file system here by the browser that would be like [http://master-IP:50070](http://master-ip:50070/). By pointing the browser to this URL, you can get the cluster information like space used / available, the number of live nodes, the number of dead nodes etc.

### HDFS File Read Operation

Whenever a client wants to read any file from HDFS, the client needs to interact with namenode as Namenode is the only place which stores metadata about data nodes. Namenode specifies the address or the location of the slaves where data is stored. The client will interact with the specified data nodes and read the data from there. For security/authentication purpose, namenode provides token to the client which it shows to the data node for reading the file.

If the client wants to read data that is stored in HDFS, it needs to interact with namenode first. So client interacts with distributed file system API and sends a request to namenode to send block location. Thus, Namenode checks if the client is having sufficient privileges to access the data or not? Then namenode will share the address at which data is stored in data node.

With the address, namenode also shares a security token with the client which it needs to show to datanode before accessing the data for authentication purpose.

When a client goes to datanode for reading the file, after checking the token, datanode allows the client to read that particular block. A client then opens input stream and starts reading data from the specified datanodes. Hence, In this manner, the client reads data directly from datanode.

If during the reading of a file datanode goes down suddenly, then a client will again go to namenode and namenode will share other location where that block is present. [Learn more about File read operation](http://data-flair.training/blogs/data-read-operation-in-hdfs/).

### 9.2. HDFS File Write Operation

As seen while reading a file, the client needs to interact with Namenode. Similarly for writing a file also client needs to interact with namenode. Hence, Namenode provides the address of the slaves on which data has to be written by the client.

Once the client finishes writing the block, the slave starts replicating the block into another slave which then copies the block to the third slave. This is the case when default replication factor of 3 is used. After required replication is created, it sends a final acknowledgment to the client. Hence, the authentication process is similar as seen in the read section.

Whenever a client needs to write any data, it needs to interact with namenode. So client interacts with distributed file system API and sends a request to namenode to send slave location.

Namenode shares the location at which data has to be written. Then client interacts with datanode at which data has to be written and starts writing the data through FS data output stream. Once the data is written and replicated, datanode sends an acknowledgment to the client informing that the data is written completely.

We have already discussed that to write a file, the client needs to interact with namenode first and then start writing data on datanodes as informed by namenode.

As soon as the client finishes writing the first block, the first datanode will copy the same block to other datanode. Thus this datanode after receiving the block, starts copying this block to the third datanode. Third sends an acknowledgment to second, the second datanode sends an acknowledgment to the first datanode and then the first datanode sends the final acknowledgment (in the case of default replication factor).

The client is sending just 1 copy of data irrespective of our replication factor while datanodes replicate the blocks. Hence, Writing of file in HDFS is not costly as parallelly multiple blocks are getting written on several datanodes. [Learn more about file write operation](http://data-flair.training/blogs/data-write-pipeline-operation-hdfs/).