**Hadoop HDFS Data Read and Write Operations**

**1. Objective**

**HDFS** follow ***Write once Read many*** models. So we cannot edit files already stored in HDFS, but we can append data by reopening the file. In Read-Write operation client first, interact with the [**NameNode**](https://data-flair.training/blogs/hadoop-hdfs-namenode-high-availability/). NameNode provides privileges so, the client can easily read and write data blocks into/from the respective datanodes. In this blog, we will discuss the internals of [**Hadoop**](https://data-flair.training/blogs/hadoop-introduction-tutorial-quick-guide/) **HDFS** data read and write operations. We will also cover how client read and write the data from [**HDFS**](https://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/), how the client interacts with master and slave nodes in HDFS data read and write operations.

Hadoop HDFS Data Read and Write Operations

This blog also contains the videos to deeply understand the internals of HDFS file read and write operations.

**2. Hadoop HDFS Data Read and Write Operations**

[**HDFS** – ***Hadoop Distributed File System***](https://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/) is the storage layer of **Hadoop**. It is most reliable storage system on the planet. HDFS works in *master-slave* fashion, **NameNode** is the master daemon which runs on the master node, **DataNode** is the slave daemon which runs on the slave node.  
Before start using with HDFS, you should install Hadoop. I recommend you-

* [**Hadoop installation on a single node**](https://data-flair.training/blogs/install-hadoop-on-single-machine/)
* [**Hadoop installation on Multi-node cluster**](https://data-flair.training/blogs/install-configure-apache-hadoop-2-7-x-on-ubuntu/)

Here, we are going to cover the HDFS data read and write operations. Let’s discuss HDFS file write operation first followed by HDFS file read operation-

**2.1. Hadoop HDFS Data Write Operation**

To [**write a file in HDFS**](https://data-flair.training/blogs/hdfs-data-write-operation/), a client needs to interact with master i.e. **namenode**(master). Now namenode provides the address of the **datanodes** (slaves) on which client will start writing the data. Client directly writes data on the datanodes, now datanode will create data write pipeline.  
The first datanode will copy the block to another datanode, which intern copy it to the third datanode. Once it creates the replicas of blocks, it sends back the acknowledgment.

**a. HDFS Data Write Pipeline Workflow**

Now let’s understand complete end to end [**HDFS**](https://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/) data write pipeline. As shown in the above figure the data write operation in HDFS is distributed, client copies the data distributedly on datanodes, the steps by step explanation of data write operation is:

**i)**The HDFS client sends a **create** request on *DistributedFileSystem* APIs.  
**ii)***DistributedFileSystem* makes an RPC call to the namenode to create a new file in the file system’s namespace.

The namenode performs various checks to make sure that the file doesn’t already exist and that the client has the permissions to create the file. When these checks pass, then only the namenode makes a record of the new file; otherwise, file creation fails and the client is thrown an *IOException.* Also Learn [**Hadoop HDFS Architecture**](https://data-flair.training/blogs/hadoop-hdfs-architecture/) in Detail.

**iii)**The *DistributedFileSystem* returns a *FSDataOutputStream* for the client to start writing data to. As the client writes data, *DFSOutputStream* splits it into packets, which it writes to an internal queue, called the data queue. The data queue is consumed by the *DataStreamer, which* is responsible for asking the namenode to allocate new [**blocks**](https://data-flair.training/blogs/data-blocks-hdfs-hadoop-distributed-file-system/) by picking a list of suitable datanodes to store the replicas.

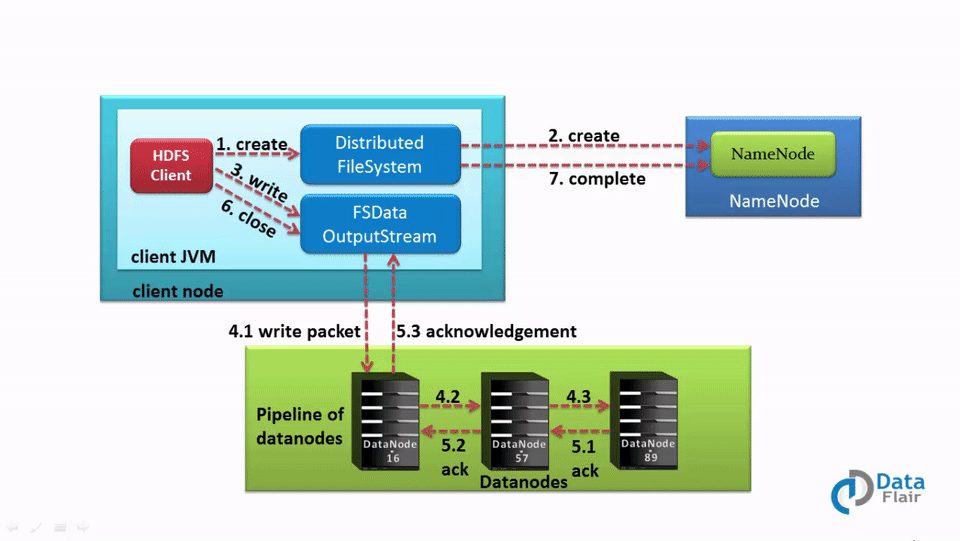
**iv)**The list of datanodes form a pipeline, and here we’ll assume the replication level is three, so there are three nodes in the pipeline. The *DataStreamer* streams the packets to the first datanode in the pipeline, which stores the packet and forwards it to the second datanode in the pipeline. Similarly, the second datanode stores the packet and forwards it to the third (and last) datanode in the pipeline. Learn [**HDFS Data blocks in detail**.](https://data-flair.training/blogs/data-blocks-hdfs-hadoop-distributed-file-system/)

**v)***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 the datanodes in the pipeline. Datanode sends the acknowledgment once required replicas are created (3 by default). Similarly, all the blocks are stored and replicated on the different datanodes, the data blocks are copied in parallel.

**vi)**When the client has finished writing data, it calls **close()** on the stream.

**vii)**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. The namenode already knows which blocks the file is made up of, so it only has to wait for blocks to be minimally replicated before returning successfully.

**Learn:** [**Hadoop HDFS Data Read and Write Operations**](https://data-flair.training/blogs/hadoop-hdfs-data-read-and-write-operations/)  
*We can summarize the HDFS data write operation from the following diagram:*

[](https://data-flair.training/blogs/wp-content/uploads/sites/2/2016/05/Data-Write-Mechanism-in-HDFS.gif)

Data Write Mechanism in HDFS Tutorial

**b. How to Write a file in HDFS – Java Program**

A sample code to write a file to HDFS in Java is as follows (To interact with HDFS and perform various operations[**follow this HDFS command part – 1**](https://data-flair.training/blogs/top-hdfs-commands-tutorial/)):

[php]FileSystem fileSystem = FileSystem.get(conf);  
// Check if the file already exists  
Path path = new Path(“/path/to/file.ext”);  
if (fileSystem.exists(path)) {  
System.out.println(“File ” + dest + ” already exists”);  
return;  
}  
// Create a new file and write data to it.  
FSDataOutputStream out = fileSystem.create(path);  
InputStream in = new BufferedInputStream(new FileInputStream(  
new File(source)));  
byte[] b = new byte[1024];  
int numBytes = 0;  
while ((numBytes = in.read(b)) > 0) {  
out.write(b, 0, numBytes);  
}  
// Close all the file descripters  
in.close();  
out.close();  
fileSystem.close();  
[/php]

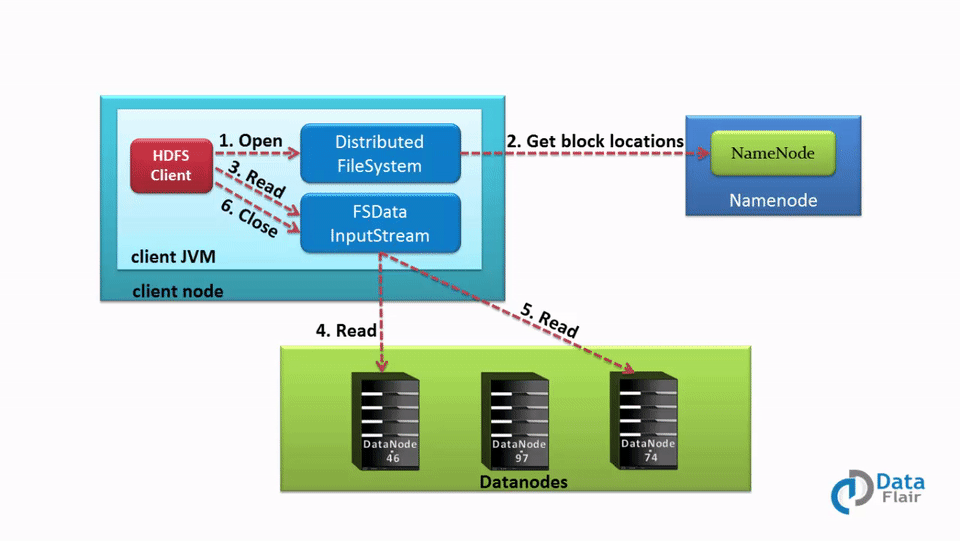
[**Hadoop HDFS Commands with Examples and Usage – Part II**](https://data-flair.training/blogs/most-used-hdfs-commands-tutorial-examples/)

**2.2. Hadoop HDFS Data Read Operation**

To [**read a file from HDFS**](https://data-flair.training/blogs/hdfs-data-read-operation/), a client needs to interact with namenode (master) as namenode is the centerpiece of [**Hadoop cluster**](https://data-flair.training/blogs/install-configure-apache-hadoop-2-7-x-on-ubuntu/) (it stores all the metadata i.e. data about the data). Now namenode checks for required privileges, if the client has sufficient privileges then namenode provides the address of the slaves where a file is stored. Now client will interact directly with the respective datanodes to read the data blocks.

**a. HDFS File Read Workflow**

Now let’s understand complete end to end HDFS data read operation. As shown in the above figure the data read operation in HDFS is distributed, the client reads the data parallelly from datanodes, the steps by step explanation of data read cycle is:

[](https://data-flair.training/blogs/wp-content/uploads/Data-Read-Mechanism-in-HDFS.gif)

HDFS data read and write operations

**i)** Client opens the file it wishes to read by calling **open()** on the *FileSystem* object, which for HDFS is an instance of *DistributedFileSystem*. See [**Data Read Operation**](https://data-flair.training/blogs/hdfs-data-read-operation/) in HDFS  
**ii)** *DistributedFileSystem* calls the namenode using RPC to determine the locations of the blocks for the first few blocks in the file. For each block, the namenode returns the addresses of the datanodes that have a copy of that [**block**](https://data-flair.training/blogs/data-blocks-hdfs-hadoop-distributed-file-system/) and datanode are sorted according to their proximity to the client.  
**iii)** *DistributedFileSystem* returns a *FSDataInputStream* to the client for it to read data from. *FSDataInputStream*, thus, wraps the *DFSInputStream* which manages the datanode and namenode I/O. Client calls **read()** on the stream. DFSInputStream which has stored the datanode addresses then connects to the closest datanode for the first block in the file.  
**iv)** Data is streamed from the datanode back to the client, as a result client can call **read()** repeatedly on the stream. When the block ends, DFSInputStream will close the connection to the datanode and then finds the best datanode for the next block. Also learn about [**Data write operation**](https://data-flair.training/blogs/hdfs-data-write-operation/) in HDFS  
**v)**If the *DFSInputStream* encounters an error while communicating with a datanode, it will try the next closest one for that block. It will also remember datanodes that have failed so that it doesn’t needlessly retry them for later blocks. The *DFSInputStream* also verifies checksums for the data transferred to it from the datanode. If it finds a corrupt block, it reports this to the namenode before the *DFSInputStream* attempts to read a replica of the block from another datanode.**vi)**When the client has finished reading the data, it calls **close()** on the stream.  
*We can summarize the HDFS data read operation from the following diagram:*

**b. How to Read a file from HDFS – Java Program**

A sample code to read a file from HDFS is as follows (To perform HDFS read and write operations [**follow this HDFS command part – 3**](https://data-flair.training/blogs/hadoop-hdfs-commands-tutorial/)):  
[php]FileSystem fileSystem = FileSystem.get(conf);  
Path path = new Path(“/path/to/file.ext”);  
if (!fileSystem.exists(path)) {  
System.out.println(“File does not exists”);  
return;  
}  
FSDataInputStream in = fileSystem.open(path);  
int numBytes = 0;  
while ((numBytes = in.read(b))> 0) {  
System.out.prinln((char)numBytes));// code to manipulate the data which is read  
}  
in.close();  
out.close();  
fileSystem.close();[/php]

**3. Fault Tolerance in HDFS**

As we have discussed HDFS data read and write operations in detail, Now, what happens when one of the machines i.e. part of the pipeline which has a datanode process running fails. Hadoop has an inbuilt functionality to handle this scenario ([**HDFS is fault tolerant**](https://data-flair.training/blogs/learn-hadoop-hdfs-fault-tolerance/)). When 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 so that datanode that are downstream from the failed node will not miss any packets.
* The current block on the good datanode is given a new identity, which is communicated to the namenode so that the partial block on the failed datanode will be deleted if the failed datanode recovery later on. Also read [**Namenode high availability**](https://data-flair.training/blogs/hadoop-hdfs-namenode-high-availability/) in HDFS
* The datanode that fails is removed from the pipeline, and then the remainder of the block’s data is written to the two good datanodes in the pipeline.
* The namenode notices that the block is under-replicated, and it arranges for a further replica to be created on another node. Then it treats the subsequent blocks as normal.

It’s possible, but unlikely, that multiple datanodes fail while client writes a block. As long as it writes *dfs.replication.min* replicas (**which default to** **1**), the write will be successful, and the block will asynchronously replicate across the cluster until it achieves target replication factor  (*dfs.replication*, **which defaults to** **3**).  
Failover process is same as data write operation, data read operation is also fault tolerant.

**Learn:** [**NameNode High Availability in Hadoop HDFS**](https://data-flair.training/blogs/hadoop-hdfs-namenode-high-availability/)

**4. Conclusion**

In conclusion, This design allows HDFS to increase the numbers of clients. This is because the data traffic spreads on all the Datanodes of clusters. It also provides [**High Availability**](https://data-flair.training/blogs/hadoop-hdfs-namenode-high-availability/), [**Rack Awareness**](https://data-flair.training/blogs/rack-awareness-hadoop-hdfs/), [**Erasure coding**](https://data-flair.training/blogs/what-is-erasure-coding-introduction-hadoop-hdfs/) etc, As a result, it empowers Hadoop.  
If you like this post or have any query about HDFS data read and write operations, so please leave a comment. We will be happy to solve them.