Before moving ahead in this HDFS tutorial blog, let me take you through some of the insane statistics related to HDFS:

* In 2010,**Facebook** claimed to have one of the largest HDFS cluster storing **21 Petabytes**of data.
* In 2012, **Facebook** declared that they have the largest single HDFS cluster with more than**100 PB**of data**.**
* And **Yahoo**! has more than **100,000 CPU** in over **40,000 servers** running Hadoop, with its biggest Hadoop cluster running**4,500 nodes**. All told, Yahoo! stores **455 petabytes**of data in HDFS.
* In fact, by 2013, most of the big names in the Fortune 50 started using Hadoop.

**DFS or Distributed File System:**

Distributed File System talks about**managing data**, i.e.**files or folders across multiple computers or servers.** In other words, DFS is a file system that allows us to store data over multiple nodes or machines in a cluster and allows multiple users to access data. So basically, it serves the same purpose as the file system which is available in your machine, like for windows you have NTFS (New Technology File System) or for Mac you have HFS (Hierarchical File System). The only difference is that, in case of Distributed File System, you store data in multiple machines rather than single machine. Even though the files are stored across the network, DFS organizes, and displays data in such a manner that a user sitting on a machine will feel like all the data is stored in that very machine.

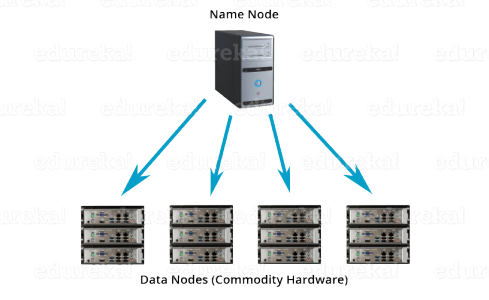
**What is HDFS?**

Hadoop Distributed file system or HDFS is a Java based distributed file system that allows you to store large data across multiple nodes in a Hadoop cluster. So, if you install Hadoop, you get HDFS as an underlying storage system for storing the data in the distributed environment.

Let’s take an example to understand it. Imagine that you have ten machines or ten computers with a hard drive of 1 TB on each machine. Now, HDFS says that if you install Hadoop as a platform on top of these ten machines, you will get HDFS as a storage service. Hadoop Distributed File System is distributed in such a way that every machine contributes their individual storage for storing any kind of data.

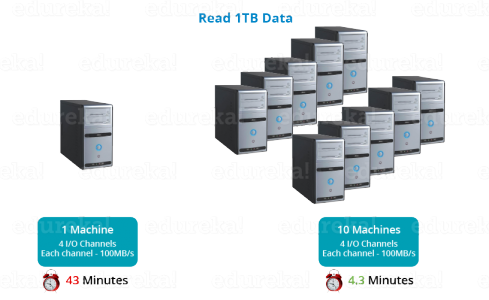
**HDFS Tutorial: Advantages Of HDFS**

1. Distributed Storage:

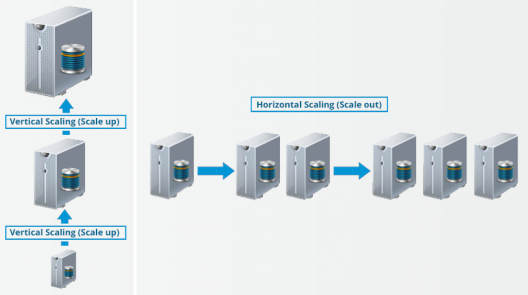


When you access Hadoop Distributed file system from any of the ten machines in the Hadoop cluster, you will feel as if you have logged into a single large machine which has a storage capacity of 10 TB (total storage over ten machines). What does it mean? It means that you can store a single large file of 10 TB which will be distributed over the ten machines (1 TB each). So, it is **not limited to the physical boundaries** of each individual machine.

2. Distributed & Parallel Computation:

Because the data is divided across the machines, it allows us to take advantage of **Distributed and Parallel Computation**. Let’s understand this concept by the above example. Suppose, it takes 43 minutes to process 1 TB file on a single machine. So, now tell me, how much time will it take to process the same 1 TB file when you have 10 machines in a Hadoop cluster with similar configuration – 43 minutes or 4.3 minutes? 4.3 minutes, Right! What happened here? Each of the nodes is working with a part of the 1 TB file in parallel. Therefore, the work which was taking 43 minutes before, gets finished in just 4.3 minutes now as the work got divided over ten machines.

3. Horizontal Scalability:



Last but not the least, let us talkabout the **horizontal scaling** or **scaling out**in Hadoop. There are two types of scaling: **vertical**and**horizontal**. In vertical scaling (scale up), you increase the hardware capacity of your system. In other words, you procure more RAM or CPU and add it to your existing system to make it more robust and powerful. But there are challenges associated with vertical scaling or scaling up:

* There is always a limit to which you can increase your hardware capacity. So, you can’t keep on increasing the RAM or CPU of the machine.
* In vertical scaling, you stop your machine first. Then you increase the RAM or CPU to make it a more robust hardware stack. After you have increased your hardware capacity, you restart the machine. This down time when you are stopping your system becomes a challenge.

In case of **horizontal scaling (scale out)**, you add more nodes to existing cluster instead of increasing the hardware capacity of individual machines. And most importantly, you can**add more machines on the go**i.e. Without stopping the system**.** Therefore, while scaling out we don’t have any down time or green zone, nothing of such sort. At the end of the day, you will have more machines working in parallel to meet your requirements.

**Features of HDFS**

We will understand these features in detail when we will explore the HDFS Architecture in our next HDFS tutorial blog. But, for now, let’s have an overview on the features of HDFS:

* ***Cost:*** The HDFS, in general, is deployed on a commodity hardware like your desktop/laptop which you use every day. So, it is very economical in terms of the cost of ownership of the project. Since, we are using low cost commodity hardware, you don’t need to spend huge amount of money for scaling out your Hadoop cluster. In other words, adding more nodes to your HDFS is cost effective.
* ***Variety and Volume of Data:*** When we talk about HDFS then we talk about storing huge data i.e. Terabytes & petabytes of data and different kinds of data. So, you can store any type of data into HDFS, be it structured, unstructured or semi structured.
* ***Reliability and Fault Tolerance:*** When you store data on HDFS, it internally divides the given data into data blocks and stores it in a distributed fashion across your Hadoop cluster. The information regarding which data block is located on which of the data nodes is recorded in the metadata. **NameNode** manages the meta data and the **DataNodes** are responsible for storing the data.  
  Name node also replicates the data i.e. maintains multiple copies of the data. This replication of the data makes HDFS very reliable and fault tolerant. So, even if any of the nodes fails, we can retrieve the data from the replicas residing on other data nodes. By default, the replication factor is 3. Therefore, if you store 1 GB of file in HDFS, it will finally occupy 3 GB of space. The name node periodically updates the metadata and maintains the replication factor consistent.
* ***Data Integrity:*** Data Integrity talks about whether the data stored in my HDFS are correct or not. HDFS constantly checks the integrity of data stored against its checksum. If it finds any fault, it reports to the name node about it. Then, the name node creates additional new replicas and therefore deletes the corrupted copies.
* ***High Throughput:***Throughput is the amount of work done in a unit time. It talks about how fast you can access the data from the file system. Basically, it gives you an insight about the system performance. As you have seen in the above example where we used ten machines collectively to enhance computation. There we were able to reduce the processing time from**43 minutes** to a mere **4.3 minutes** as all the machines were working in parallel. Therefore, by processing data in parallel, we decreased the processing time tremendously and thus, achieved high throughput.
* ***Data Locality:***Data locality talks about moving processing unit to data rather than the data to processing unit. In our traditional system, we used to bring the data to the application layer and then process it. But now, because of the architecture and huge volume of the data, bringing the data to the application layer will reduce the network performance to a noticeable extent. So, in HDFS, we bring the computation part to the data nodes where the data is residing. Hence, you are not moving the data, you are bringing the program or processing part to the data.

## ****Apache Hadoop HDFS**** ****Architecture****

**Apache Hadoop HDFS Architecture**

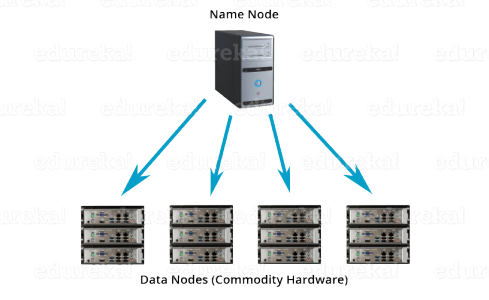
**Introduction:**

In this blog, I am going to talk about Apache Hadoop HDFS Architecture. From my [***previous blog***](http://www.edureka.co/blog/hdfs-tutorial/?utm_source=blog&utm_medium=content-link&utm_campaign=apache-hadoop-hdfs-architecture), you already know that HDFS is a distributed file system which is deployed on low cost commodity hardware. So, it’s high time that we should take a deep dive into Apache Hadoop HDFS Architecture and unlock its beauty.

The topics that will be covered in this blog on Apache Hadoop HDFS Architecture are as following:

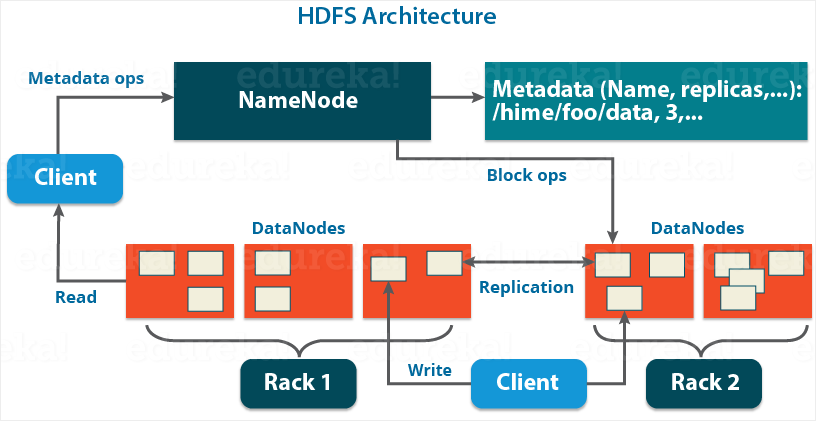
* [**HDFS Master/Slave Topology**](https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/?utm_source=blog&utm_medium=left-menu&utm_term=Apache%20Hadoop%20HDFS%20Architecture#hdfs_architecture)
* [**NameNode**](https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/?utm_source=blog&utm_medium=left-menu&utm_term=Apache%20Hadoop%20HDFS%20Architecture#namenode)**, [DataNode](https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/?utm_source=blog&utm_medium=left-menu&utm_term=Apache%20Hadoop%20HDFS%20Architecture" \l "datanode) and**[**Secondary NameNode**](https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/?utm_source=blog&utm_medium=left-menu&utm_term=Apache%20Hadoop%20HDFS%20Architecture#secondary_namenode)
* [**What is a block?**](https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/?utm_source=blog&utm_medium=left-menu&utm_term=Apache%20Hadoop%20HDFS%20Architecture#hdfs_block)
* [**Replication Management**](https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/?utm_source=blog&utm_medium=left-menu&utm_term=Apache%20Hadoop%20HDFS%20Architecture#replication_management)
* [**Rack Awareness**](https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/?utm_source=blog&utm_medium=left-menu&utm_term=Apache%20Hadoop%20HDFS%20Architecture#rack_awareness)
* [**HDFS Read/Write – Behind the scenes**](https://www.edureka.co/blog/apache-hadoop-hdfs-architecture/?utm_source=blog&utm_medium=left-menu&utm_term=Apache%20Hadoop%20HDFS%20Architecture#hdfs_read_or_write_architecture)

**HDFS Architecture:**



**Apache HDFS** or **Hadoop Distributed File System** is a block-structured file system where each file is divided into blocks of a pre-determined size. These blocks are stored across a cluster of one or several machines. Apache Hadoop HDFS Architecture follows a *Master/Slave Architecture*, where a cluster comprises of a single NameNode (Master node) and all the other nodes are DataNodes (Slave nodes). HDFS can be deployed on a broad spectrum of machines that support Java. Though one can run several DataNodes on a single machine, but in the practical world, these DataNodes are spread across various machines.

**NameNode:**



NameNode is the master node in the Apache Hadoop HDFS Architecture that maintains and manages the blocks present on the DataNodes (slave nodes). NameNode is a very highly available server that manages the File System Namespace and controls access to files by clients. I will be discussing this High Availability feature of Apache Hadoop HDFS in my next blog. The HDFS architecture is built in such a way that the user data never resides on the NameNode. The data resides on DataNodes only.

*Functions of NameNode:*

* It is the master daemon that maintains and manages the DataNodes (slave nodes)
* It records the metadata of all the files stored in the cluster, e.g. The location of blocks stored, the size of the files, permissions, hierarchy, etc. There are two files associated with the metadata:
  + **FsImage:** It contains the complete state of the file system namespace since the start of the NameNode.
  + **EditLogs:** It contains all the recent modifications made to the file system with respect to the most recent FsImage.
* It records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog.
* It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live.
* It keeps a record of all the blocks in HDFS and in which nodes these blocks are located.
* The NameNode is also responsible to take care of the **replication factor**of all the blocks which we will discuss in detail later in this HDFS tutorial blog.
* In **case of the DataNode failure**, the NameNode chooses new DataNodes for new replicas, balance disk usage and manages the communication traffic to the DataNodes.

**DataNode:**

DataNodes are the slave nodes in HDFS. Unlike NameNode, DataNode is a commodity hardware, that is, a non-expensive system which is not of high quality or high-availability. The DataNode is a block server that stores the data in the local file ext3 or ext4.

*Functions of DataNode:*

* These are slave daemons or process which runs on each slave machine.
* The actual data is stored on DataNodes.
* The DataNodes perform the low-level read and write requests from the file system’s clients.
* They send heartbeats to the NameNode periodically to report the overall health of HDFS, by default, this frequency is set to 3 seconds.

Till now, you must have realized that the NameNode is pretty much important to us. If it fails, we are doomed.  But don’t worry, we will be talking about how Hadoop solved this single point of failure problem in the next Apache Hadoop HDFS Architecture blog. So, just relax for now and let’s take one step at a time.

**Secondary NameNode:**

Apart from these two daemons, there is a third daemon or a process called Secondary NameNode. The Secondary NameNode works concurrently with the primary NameNode as a **helper daemon.**And don’t be confused about the Secondary NameNode being a**backup NameNode because it is not.**



*Functions of Secondary NameNode:*

* The Secondary NameNode is one which constantly reads all the file systems and metadata from the RAM of the NameNode and writes it into the hard disk or the file system.
* It is responsible for combining the EditLogswith FsImage from the NameNode.
* It downloads the EditLogs from the NameNode at regular intervals and applies to FsImage. The new FsImage is copied back to the NameNode, which is used whenever the NameNode is started the next time.

Hence, Secondary NameNode performs regular checkpoints in HDFS. Therefore, it is also called CheckpointNode.

[**Become a HDFS Expert!**](http://www.edureka.co/big-data-and-hadoop?utm_source=blog&utm_medium=blog-cta&utm_campaign=apache-hadoop-hdfs-architecture-1)

**Blocks:**

Now, as we know that the data in HDFS is scattered across the DataNodes as blocks. **Let’s have a look at what is a block and how is it formed?**

Blocks are the nothing but the smallest continuous location on your hard drive where data is stored. In general, in any of the File System, you store the data as a collection of blocks. Similarly, HDFS stores each file as blocks which are scattered throughout the Apache Hadoop cluster. The default size of each block is 128 MB in Apache Hadoop 2.x (64 MB in Apache Hadoop 1.x) which you can configure as per your requirement.

HDFS ARCHITECTURE – HDFS TUTORIAL Introduction In this blog, we are going to talk about HDFS Architecture. From my previous blog, we already know that HDFS is a distributed file system which is deployed on low cost commodity hardware. I discussed many of its features too. So, its high time that we take a deep dive into Apache Hadoop HDFS Architecture and unlock its beauty. The topics that will be covered in this blog are as follows: • HDFS Master/Slave Topology • NameNode and DataNode • What is a block • Replication Management • Rack Awareness • HDFS Read/Write – Behind the scenes HDFS Architecture HDFS or Hadoop Distributed File System is a block-structured file system where each file is divided into blocks of a pre-determined size. These blocks are stored across a cluster of one or several machines. Apache Hadoop HDFS Architecture follows a Master/Slave Architecture, where a cluster comprises of a single NameNode (Master node) and all the other nodes are DataNodes (Slave nodes). HDFS is based on Java programming language, due to which HDFS can be deployed on broad spectrum of machines that support Java. Though one can run several DataNodes on a single machine, but in practical world, these DataNodes are spread across various machines. NameNode and DataNode NameNode: NameNode is the master of HDFS that maintains and manages the blocks present on the DataNodes (slave nodes). Think of the NameNode as a Lamborghini in midst of various other cars. Thus, like a Lamborghini, NameNode is a very highly available server that manages the File System Namespace and controls access to files by clients. I will be discussing this High Availability feature of Apache Hadoop HDFS in my next blog. The HDFS architecture is built in such a way that the user data is never stored in the NameNode. The data resides on DataNodes only. Functions of NameNode: • It is the master daemon that maintains and manages the DataNodes (slave nodes) • It records the metadata of all the files stored in the cluster, e.g. location of blocks stored, size of the files, permissions, hierarchy, etc. There are two files associated with metadata: o FsImage: An image of the file system on starting the NameNode. o EditLogs: A series of modifications done to the file system after starting the NameNode. • It records each change that takes place to the file system metadata. For example, if a file is deleted in HDFS, the NameNode will immediately record this in the EditLog. • It regularly receives a Heartbeat and a block report from all the DataNodes in the cluster to ensure that the DataNodes are live. • It keeps a record of all the blocks in HDFS and in which nodes these blocks are located. • The NameNode is also responsible to take care of the replication factor of all the blocks which we will discuss in detail later in this HDFS tutorial blog. • In case of a DataNode failure, the NameNode chooses new DataNodes for new replicas, balances disk usage and manages the communication traffic to the DataNodes. DataNode: DataNodes are the slave nodes in HDFS, just like any average car in front of a Lamborghini! Unlike NameNode, DataNode is a commodity hardware, that is, a non-expensive system which is not of high quality or high-availability. DataNode is a block server that stores the data in the local file ext3 or ext4. Functions of DataNode: • These are slave daemons or process which runs on each slave machine. • The actual data is stored on DataNodes. • DataNodes perform the low-level read and write requests from the file system’s clients. • They send heartbeats to the NameNode periodically to report the overall health of HDFS, by default, this frequency is set to 3 seconds. So, till now, you folks must have realized that the NameNode is pretty much important to us. If it fails, we are doomed. But don’t worry, we will be talking about how Hadoop solved this single point of failure problem in the next HDFS tutorial blog. So, just relax for now and let’s take one step at a time. Secondary NameNode Apart from these two daemons there is a third daemon or process called Secondary NameNode. The Secondary NameNode works concurrently with the primary NameNode as a helper daemon. And don’t confuse Secondary NameNode as a backup NameNode because it is not. Functions of Secondary NameNode: • The Secondary NameNode is one which constantly reads all the file systems and metadata from the RAM of the NameNode and writes it into the hard disk or the file system. • It is responsible for combining the EditLogs with FsImage from the NameNode. • It downloads the EditLogs from the NameNode at regular intervals and applies to FsImage. The new FsImage is copied back to the NameNode, which is used whenever the NameNode is started the next time. Hence, Secondary NameNode just performs regular checkpoints in HDFS. Therefore, it is also called CheckpointNode. Blocks Now as we know that the data in HDFS is scattered across the DataNodes as blocks. Let’s have a look on what is a block and how is it formed? Blocks are the nothing but smallest continuous location in your hard drive where data is stored. In general, in any of the File System the data are stored as collection of blocks. Similarly, HDFS stores each file as blocks which is scattered throughout the Apache Hadoop cluster. The default size of each block is 128MB in Apache Hadoop 2.x (64 MB in Apache Hadoop 1.x) which you can configure as per your requirement. It is not necessary that in HDFS, each file is stored in exact multiple of the configured block size (128MB, 256MB etc.). Let’s take an example where I have a file “example.txt” of size 514MB as shown in above figure. Suppose, we are using the default block size configuration which is 128Mb. So, how many blocks will be created? 5, Right. First four blocks will be of 128 MB. But, the last block will be of 2 MB size only. Now you must be thinking why we need to have such a huge blocks size i.e. 128MB? Well, whenever we talk about HDFS, we talk about huge data sets i.e. terabytes and petabytes of data. So, if we had a block size of let’s say 4KB as in Linux file system, we would be having too many of blocks and therefore too much of metadata. So, managing these no. of blocks and metadata will create huge overhead which is something, we don’t want. As you understood what a block is, lets understand how these blocks are places in the next section. Replication Management and Rack Awareness Replication Management: HDFS provides a reliable way to store huge data in a distributed environment as data blocks. The blocks are also replicated to provide fault tolerance. The default replication factor is 3 which is again configurable. So, if you want to store a file of 1GB in your HDFS, you will be consuming a space of 3GB (replication factor =3) and 24 (1GB/128MB=8 data blocks) data blocks in total, considering the default configuration. Don’t worry guys, if you didn’t get the math in one go. Take your time then move ahead. The NameNode collects block report from DataNode periodically to maintain the replication factor. Therefore, whenever a block is over-replicated or under-replicated the NameNode deletes or add replicas as needed. Rack Awareness: Anyways, moving ahead, let’s talk more about how replica are placed and what is rack awareness? Again, the NameNode also ensures that all the replicas are not stored on the same rack or a single rack. It follows an in-built Rack Awareness Algorithm to reduce latency. Considering the replication factor is 3, the Rack Awareness Algorithm says that the first replica of a block will be stored on a local rack and the next two replicas will be stored on a different (remote) rack but on a different DataNode within that (remote) rack. If you have more replicas, the rest of the replicas will be placed on random DataNodes provided not more than two replicas reside on the same rack, if possible. This is how an actual Hadoop production cluster looks like. Here, you have multiple racks populated with many DataNodes. Advantages of Rack Awareness: So now you will be thinking why do we need Rack Awareness algorithm? The reasons are: • To improve the network performance: The communication between nodes residing on different racks is directed via switch. In general, you will find greater network bandwidth between machines in the same rack than the machines residing in different rack. So, the Rack Awareness helps you to have reduce write traffic in between different racks and thus providing a better write performance. Also, you will be gaining increased read performance because you are using the bandwidth of multiple racks. • To prevent loss of data: We don’t have to worry about the data even if an entire rack fails because of the switch failure or power failure. And if you think about it, it will make sense, as it is said that never put all your eggs in the same basket. HDFS Read/ Write Architecture Now let’s talk about how the data read/write operations are performed on HDFS. HDFS follows Write Once - Read Many philosophy. So, you can’t edit files already stored in HDFS. But, you can append new data in a file. HDFS Write Architecture: Imagine a situation where a HDFS client, want to write a file named “example.txt” of size 248MB. Assume that the system block size is configured to 128 MB (default). So, the client will be dividing the file “example.txt” into 2 blocks – one of 128MB (Block A) and the other of 120 MB (block B). Now, the following protocol will be followed whenever the data is written into HDFS: • At first, the HDFS client will reach out to the NameNode for a Write Request against the two blocks, say, Block A & Block B. • The NameNode will then grant client the write permission and will provide the IP addresses of the DataNodes where the file blocks will be copied eventually. • The selection of IP addresses of DataNodes is purely randomized based on availability, replication factor and rack awareness that we have discussed earlier. • Let’s say the replication factor is set to default i.e. 3. Therefore, for each block the NameNode will be providing the client a list of (3) IP addresses of DataNodes. The list will be unique for each block. • Suppose, the NameNode provided following lists of IP addresses to the client: o For Block A, list A = {IP of DataNode 1, IP of DataNode 4, IP of DataNode 5} o For Block B, set B = {IP of DataNode 3, IP of DataNode 7, IP of DataNode 9} • Each block will be copied in three different DataNodes to maintain the replication factor consistent. • Now the whole data copy process will happen in three stages: 1. Set up of Pipeline 2. Data streaming and replication 3. Shutdown of Pipeline (Acknowledgement stage) 1. Set up of Pipeline: Before writing the blocks, client confirms whether the DataNodes present in each of the list of IPs are ready to receive the data or not. For doing so, the client creates a pipeline for each of the blocks by connecting the individual DataNodes in the respective list for that block. Let us consider Block A. The list of DataNodes provided by the NameNode is : For Block A, list A = {IP of DataNode 1, IP of DataNode 4, IP of DataNode 5}. So, for block A, the client will be performing following steps to create a pipeline:  Client will choose first DataNode in the list (DataNode IPs for Block A) which is DataNode 1 and will open a TCP/IP connection.  Client will inform DataNode 1 to be ready to receive the block. It will also provide the IPs of next two DataNodes (4 and 5) where the block will be replicated.  DataNode 1 will connect to DataNode 4. DataNode 1 will inform DataNode 4 to be ready to receive the block and will give it the IP of DataNode 5. Then, DataNode 4 will tell DataNode 5 to be ready for receiving the data.  Next, the acknowledgement of readiness will follow the reverse sequence i.e. From DataNode 5 to 4 and then to 1.  At last DataNode 1 will inform the client that all the DataNodes are ready and a pipeline is formed between client, DataNode 1, 4 and 5.  Now pipeline set up is complete and the client will finally begin the data copy or streaming process. 2. Data Streaming: As the pipeline has been created, the client will push the data into the pipeline. Now, don’t forget that in HDFS, data is replicated based on replication factor. So, here Block A will be stored to three DataNodes as the assumed replication factor is 3. Moving ahead, the client will copy the block (A) to DataNode 1 only. The replication is always done by DataNodes sequentially. So, following steps will take place during replication:  Once the block has been written to DataNode 1, it will connect to DataNode 4.  Then, it will push the block in the pipeline and data will be copied to DataNode 4.  Again, DataNode 4 will connect to DataNode 5 and will copy the block. 3. Shutdown of Pipeline or Acknowledgement stage: Once the block has been copied into all the three DataNodes, a series of acknowledgements will take place to ensure the client and NameNode that the data has been written successfully. And client will finally close the pipeline to end the TCP session. As shown in the figure, the acknowledgement happens in the reverse sequence i.e. from DataNode 5 to 4 and then to 1. Finally, the DataNode 1 will push three acknowledgements (including its own) into the pipeline and send it to client. Client will inform NameNode that data has been written successfully. NameNode will update its metadata and the client will shut down the pipeline. Similarly, Block B will also be copied into the DataNodes in parallel with Block A. But, following things are to be noticed here:  Client will copy Block A and Block B to the first DataNode simultaneously.  Therefore, in our case two pipelines will be formed for each of the block and all the process discussed above will happen in parallel in these two pipelines.  Client writes the block into first DataNode and then the DataNodes will be replicating the block sequentially. Folks! Its time for a quiz now: In HDFS, blocks of a file are written in parallel, however the replication of the blocks are done sequentially: a. True b. False True. Right guys. So, if you are still confused, go through the HDFS write architecture again. I am sure you will understand it this time. HDFS Read Architecture: HDFS Read architecture is comparatively easy to understand. Let’s the above example again where a HDFS clients want to read the file “example.txt” which is already there in HDFS. Now, following steps will be taking place while reading the file: • Client will reach out to NameNode asking for the block metadata for the file example.txt. • The NameNode will return the list of DataNodes where each block (Block A and B) are stored. • After that client, will connect to the DataNodes where the blocks are stored. • Client starts reading data parallel from the Data nodes (Block A from DataNode 1 and Block B from DataNode 5). • Once it gets all the required file blocks, it will combine these blocks to form a file. While serving read request of the client, HDFS selects the replica which is closest to the client. This reduces the read latency and the bandwidth consumption. Therefore, that replica is selected which resides on the same rack as the reader node, if possible. 

It is not necessary that in HDFS, each file is stored in exact multiple of the configured block size (128 MB, 256 MB etc.). Let’s take an example where I have a file “example.txt” of size 514 MB as shown in above figure.  Suppose that we are using the default configuration of block size, which is 128 MB. Then, how many blocks will be created? 5, Right. The first four blocks will be of 128 MB. But, the last block will be of 2 MB size only.

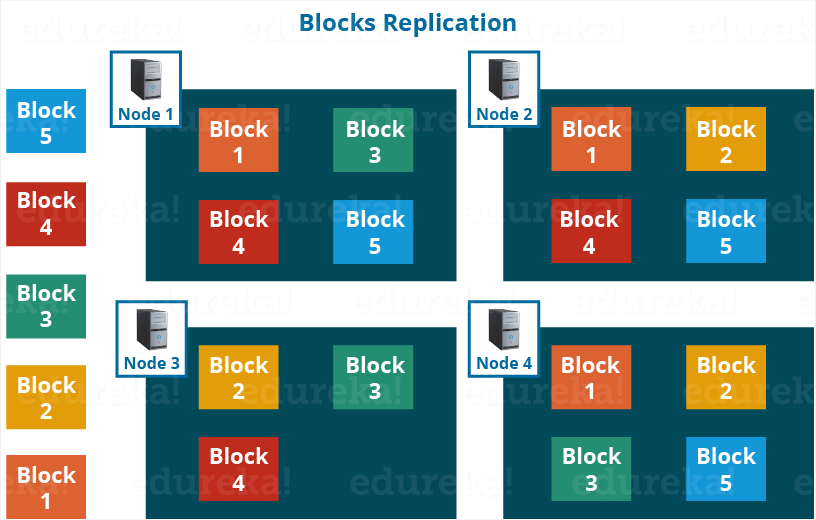
**Now, you must be thinking why we need to have such a huge blocks size i.e. 128 MB?**

Well, whenever we talk about HDFS, we talk about huge data sets, i.e. Terabytes and Petabytes of data. So, if we had a block size of let’s say of 4 KB, as in Linux file system, we would be having too many blocks and therefore too much of the metadata. So, managing these no. of blocks and metadata will create huge overhead, which is something, we don’t want.

*As you understood****what a block is****, let us understand how the replication of these blocks takes place in the next section of this HDFS Architecture. Meanwhile, you may check out this video tutorial on HDFS Architecture where all the HDFS Architecture concepts has been discussed in detail:*

**HDFS Architecture Tutorial Video | Edureka**

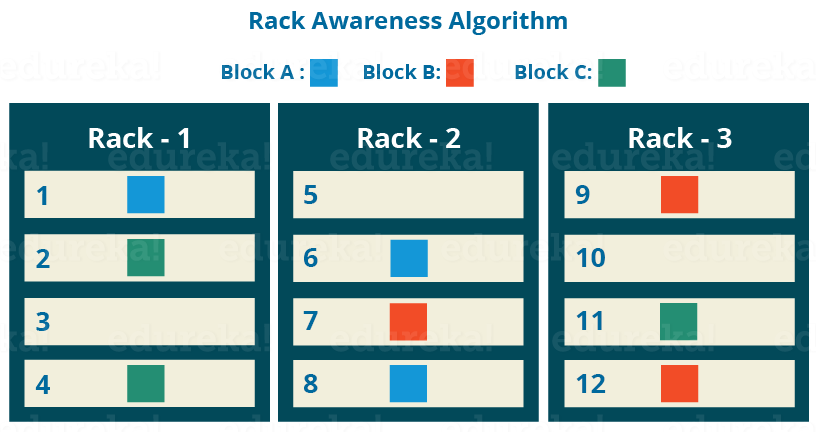
**Replication Management:**

HDFS provides a reliable way to store huge data in a distributed environment as data blocks. The blocks are also replicated to provide fault tolerance. The default replication factor is 3 which is again configurable. So, as you can see in the figure below where each block is replicated three times and stored on different DataNodes (considering the default replication factor): 

Therefore, if you are storing a file of 128 MB in HDFS using the default configuration, you will end up occupying a space of 384 MB (3\*128 MB) as the blocks will be replicated three times and each replica will be residing on a different DataNode. 

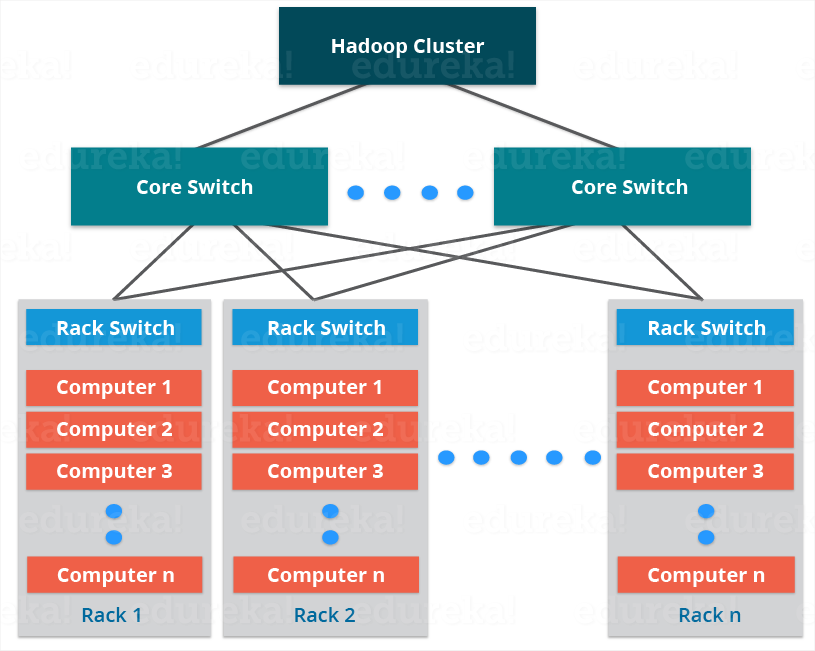
***Note:*** The NameNode collects block report from DataNode periodically to maintain the replication factor. Therefore, whenever a block is over-replicated or under-replicated the NameNode deletes or add replicas as needed.

**Rack Awareness:**



Anyways, moving ahead, let’s talk more about how HDFS places replica and what is rack awareness? Again, the NameNode also ensures that all the replicas are not stored on the same rack or a single rack. It follows an in-built Rack Awareness Algorithm to reduce latency as well as provide fault tolerance. Considering the replication factor is 3, the Rack Awareness Algorithm says that the first replica of a block will be stored on a local rack and the next two replicas will be stored on a different (remote) rack but, on a different DataNode within that (remote) rack as shown in the figure above. If you have more replicas, the rest of the replicas will be placed on random DataNodes provided not more than two replicas reside on the same rack, if possible.

This is how an actual Hadoop production cluster looks like. Here, you have multiple racks populated with DataNodes:



Advantages of Rack Awareness:

So, now you will be thinking why do we need a Rack Awareness algorithm? The reasons are:

* **To improve the network performance:**The communication between nodes residing on different racks is directed via switch. In general, you will find*greater network bandwidth* between machines in the same rack than the machines residing in different rack. So, the Rack Awareness helps you to have reduce write traffic in between different racks and thus providing a better write performance. Also, you will be gaining increased read performance because you are using the bandwidth of multiple racks.
* **To prevent loss of data:**We don’t have to worry about the data even if an entire rack fails because of the switch failure or power failure. And if you think about it, it will make sense, as it is said that *never put all your eggs in the same basket.*

[**Check out our Hadoop Course**](http://www.edureka.co/big-data-and-hadoop?utm_source=blog&utm_medium=blog-cta&utm_campaign=apache-hadoop-hdfs-architecture-1)

**HDFS Read/ Write Architecture:**

Now let’s talk about how the data read/write operations are performed on HDFS. HDFS follows Write Once – Read Many Philosophy. So, you can’t edit files already stored in HDFS. But, you can append new data by re-opening the file.

**HDFS Write Architecture:**

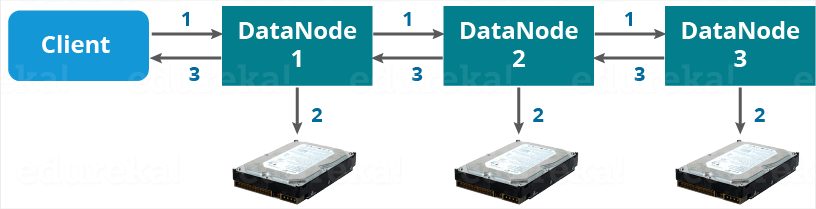
Suppose a situation where an HDFS client, wants to write a file named “example.txt” of size 248 MB.



Assume that the system block size is configured for 128 MB (default). So, the client will be dividing the file “example.txt” into 2 blocks – one of 128 MB (Block A) and the other of 120 MB (block B).

Now, the following protocol will be followed whenever the data is written into HDFS:

* At first, the HDFS client will reach out to the NameNode for a Write Request against the two blocks, say, Block A & Block B.
* The NameNode will then grant the client the write permission and will provide the IP addresses of the DataNodes where the file blocks will be copied eventually.
* The selection of IP addresses of DataNodes is purely randomized based on availability, replication factor and rack awareness that we have discussed earlier.
* Let’s say the replication factor is set to default i.e. 3. Therefore, for each block the NameNode will be providing the client a list of (3) IP addresses of DataNodes. The list will be unique for each block.
* Suppose, the NameNode provided following lists of IP addresses to the client:
  + For Block A, list A = {IP of DataNode 1, IP of DataNode 4, IP of DataNode 6}
  + For Block B, set B = {IP of DataNode 3, IP of DataNode 7, IP of DataNode 9}
* Each block will be copied in three different DataNodes to maintain the replication factor consistent throughout the cluster.
* Now the whole data copy process will happen in three stages:

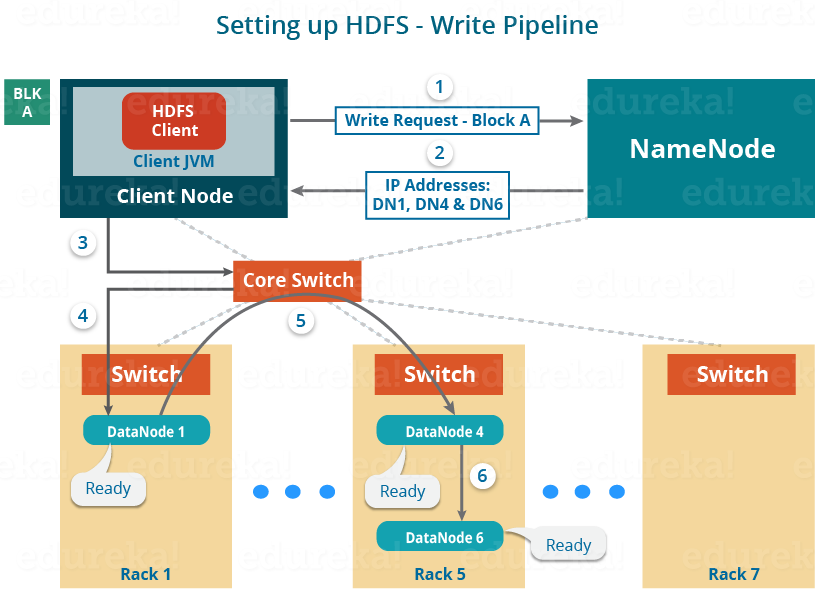


1. Set up of Pipeline
2. Data streaming and replication
3. Shutdown of Pipeline (Acknowledgement stage)

**1. Set up of Pipeline:**

Before writing the blocks, the client confirms whether the DataNodes, present in each of the list of IPs, are ready to receive the data or not. In doing so, the client creates a pipeline for each of the blocks by connecting the individual DataNodes in the respective list for that block. Let us consider Block A. The list of DataNodes provided by the NameNode is:

**For Block A, list A = {IP of DataNode 1, IP of DataNode 4, IP of DataNode 6}.**

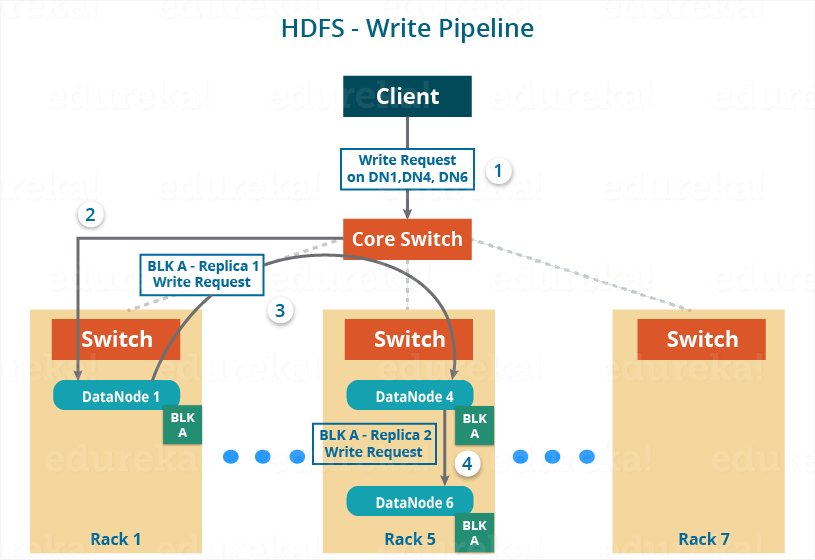


So, for block A, the client will be performing the following steps to create a pipeline:

* The client will choose the first DataNode in the list (DataNode IPs for Block A) which is DataNode 1 and will establish a TCP/IP connection.
* The client will inform DataNode 1 to be ready to receive the block. It will also provide the IPs of next two DataNodes (4 and 6) to the DataNode 1 where the block is supposed to be replicated.
* The DataNode 1 will connect to DataNode 4. The DataNode 1 will inform DataNode 4 to be ready to receive the block and will give it the IP of DataNode 6. Then, DataNode 4 will tell DataNode 6 to be ready for receiving the data.
* Next, the acknowledgement of readiness will follow the reverse sequence, i.e. From the DataNode 6 to 4 and then to 1.
* At last DataNode 1 will inform the client that all the DataNodes are ready and a pipeline will be formed between the client, DataNode 1, 4 and 6.
* Now pipeline set up is complete and the client will finally begin the data copy or streaming process.

**2. Data Streaming:**

As the pipeline has been created, the client will push the data into the pipeline. Now, don’t forget that in HDFS, data is replicated based on replication factor. So, here Block A will be stored to three DataNodes as the assumed replication factor is 3. Moving ahead, the client will copy the block (A) to DataNode 1 only. The replication is always done by DataNodes sequentially.



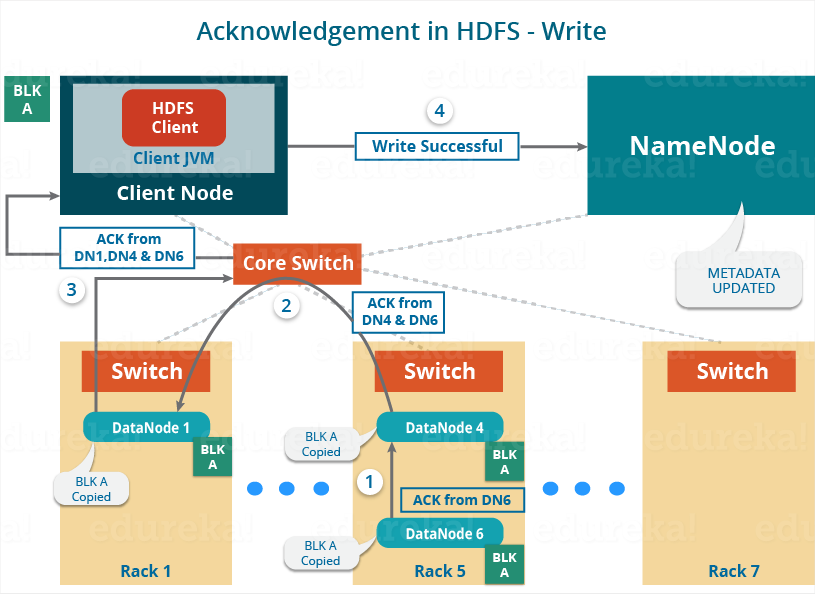
So, the following steps will take place during replication:

* Once the block has been written to DataNode 1 by the client, DataNode 1 will connect to DataNode 4.
* Then, DataNode 1 will push the block in the pipeline and data will be copied to DataNode 4.
* Again, DataNode 4 will connect to DataNode 6 and will copy the last replica of the block.

**3. Shutdown of Pipeline or Acknowledgement stage:**

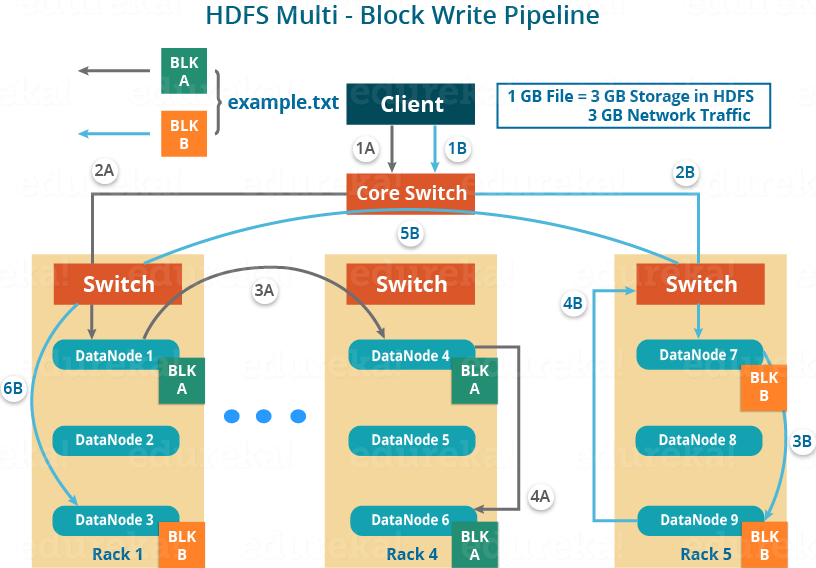
Once the block has been copied into all the three DataNodes, a series of acknowledgements will take place to ensure the client and NameNode that the data has been written successfully. Then, the client will finally close the pipeline to end the TCP session.

As shown in the figure below, the acknowledgement happens in the reverse sequence i.e. from DataNode 6 to 4 and then to 1. Finally, the DataNode 1 will push three acknowledgements (including its own) into the pipeline and send it to the client. The client will inform NameNode that data has been written successfully. The NameNode will update its metadata and the client will shut down the pipeline.



Similarly, Block B will also be copied into the DataNodes in parallel with Block A. So, the following things are to be noticed here:

* The client will copy Block A and Block B to the first DataNode **simultaneously**.
* Therefore, in our case, two pipelines will be formed for each of the block and all the process discussed above will happen in parallel in these two pipelines.
* The client writes the block into the first DataNode and then the DataNodes will be replicating the block sequentially.

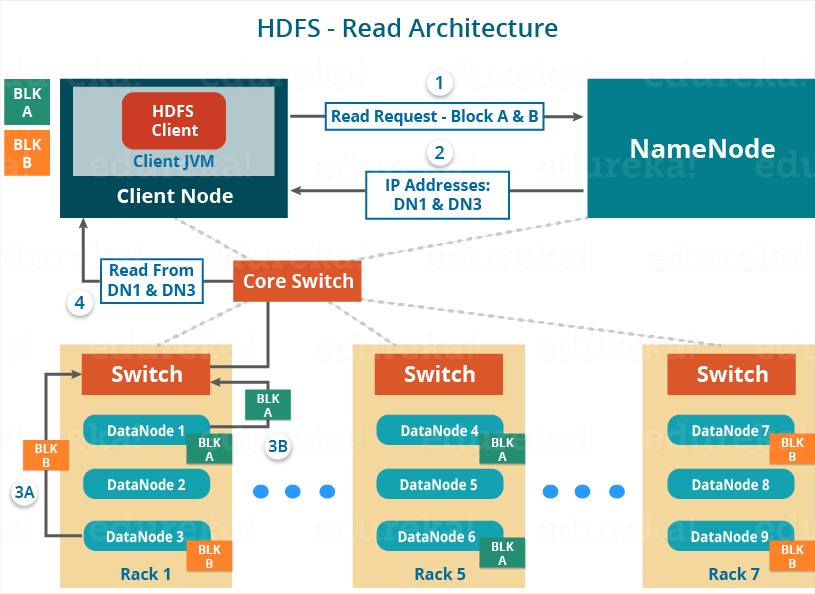


As you can see in the above image, there are two pipelines formed for each block (A and B). Following is the flow of operations that is taking place for each block in their respective pipelines:

* For Block A: 1A -> 2A -> 3A -> 4A
* For Block B: 1B -> 2B -> 3B -> 4B -> 5B -> 6B

**HDFS Read Architecture:**

HDFS Read architecture is comparatively easy to understand. Let’s take the above example again where the HDFS client wants to read the file “example.txt” now.



Now, following steps will be taking place while reading the file:

* The client will reach out to NameNode asking for the block metadata for the file “example.txt”.
* The NameNode will return the list of DataNodes where each block (Block A and B) are stored.
* After that client, will connect to the DataNodes where the blocks are stored.
* The client starts reading data parallel from the DataNodes (Block A from DataNode 1 and Block B from DataNode 3).
* Once the client gets all the required file blocks, it will combine these blocks to form a file.

While serving read request of the client, HDFS selects the replica which is closest to the client. This reduces the read latency and the bandwidth consumption. Therefore, that replica is selected which resides on the same rack as the reader node, if possible.

Now, you should have a pretty good idea about Apache Hadoop HDFS Architecture. I understand that there is a lot of information here and it may not be easy to get it in one go. I would suggest you to go through it again and I am sure you will find it easier this time. Now, in my next blog, I will be talking about Apache Hadoop HDFS Federation and High Availability Architecture.