Hadoop is an open source framework. It is provided by Apache to process and analyze very huge volume of data. It is written in Java and currently used by Google, Facebook, LinkedIn, Yahoo, Twitter etc.

Hadoop is written in Java and is not OLAP (online analytical processing). It is used for batch/offline processing.

Modules of Hadoop

1. **HDFS:** Hadoop Distributed File System. Google published its paper GFS and on the basis of that HDFS was developed. It states that the files will be broken into blocks and stored in nodes over the distributed architecture.
2. **Yarn:** Yet another Resource Negotiator is used for job scheduling and manage the cluster.
3. **Map Reduce:** This is a framework which helps Java programs to do the parallel computation on data using key value pair. The Map task takes input data and converts it into a data set which can be computed in Key value pair. The output of Map task is consumed by reduce task and then the out of reducer gives the desired result.

Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is a distributed file system for Hadoop. It contains a master/slave architecture. This architecture consist of a single NameNode performs the role of master, and multiple DataNodes performs the role of a slave.

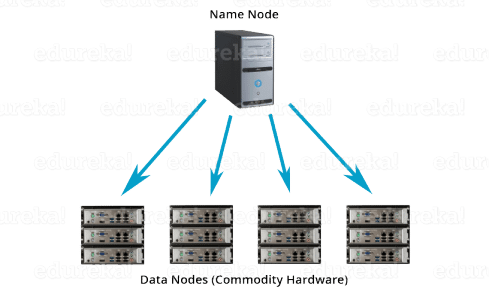
Both NameNode and DataNode are capable enough to run on commodity machines. The Java language is used to develop HDFS. So any machine that supports Java language can easily run the NameNode and DataNode software.

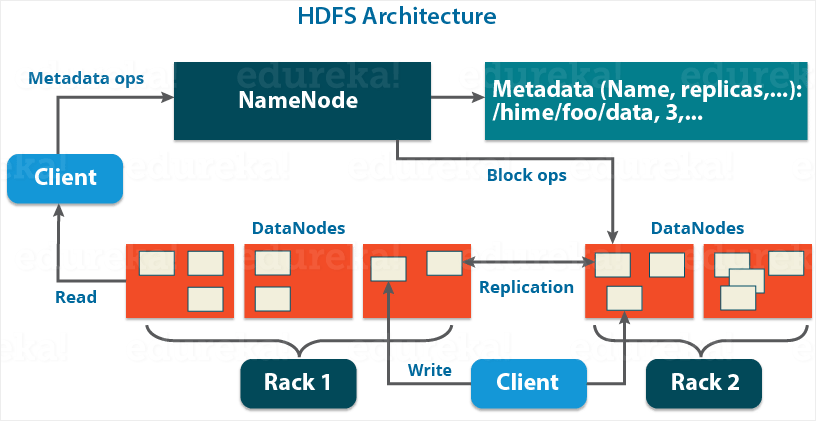
NameNode

* It is a single master server exist in the HDFS cluster.
* As it is a single node, it may become the reason of single point failure.
* It manages the file system namespace by executing an operation like the opening, renaming and closing the files.
* It simplifies the architecture of the system.

DataNode

* The HDFS cluster contains multiple DataNodes.
* Each DataNode contains multiple data blocks.
* These data blocks are used to store data.
* It is the responsibility of DataNode to read and write requests from the file system's clients.
* It performs block creation, deletion, and replication upon instruction from the NameNode.





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.

**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.

**Blocks:**

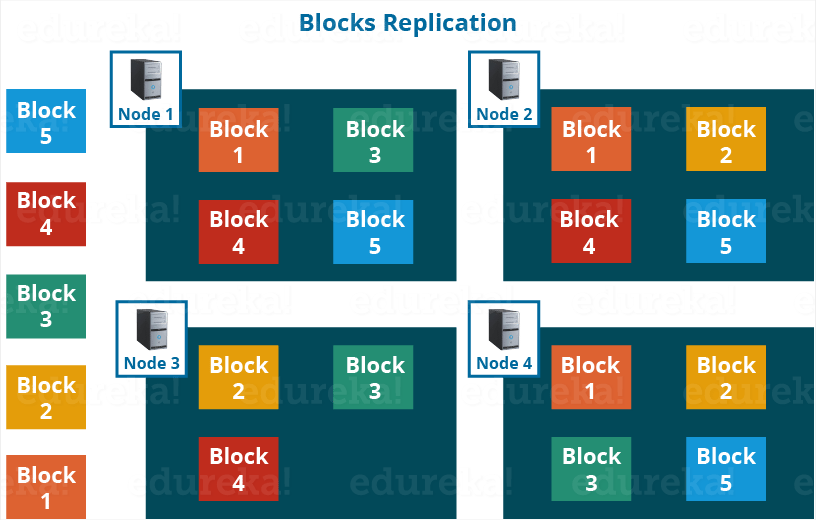
Now, as we know that the data in HDFS is scattered across the DataNodes as blocks. **Let’s have a look at what is a block and how is it formed?**

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 src=

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.

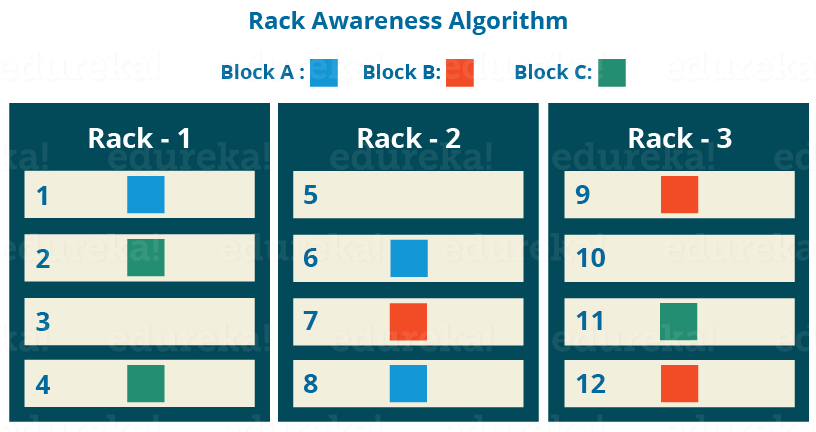
**Replication Management:**

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.

**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.

