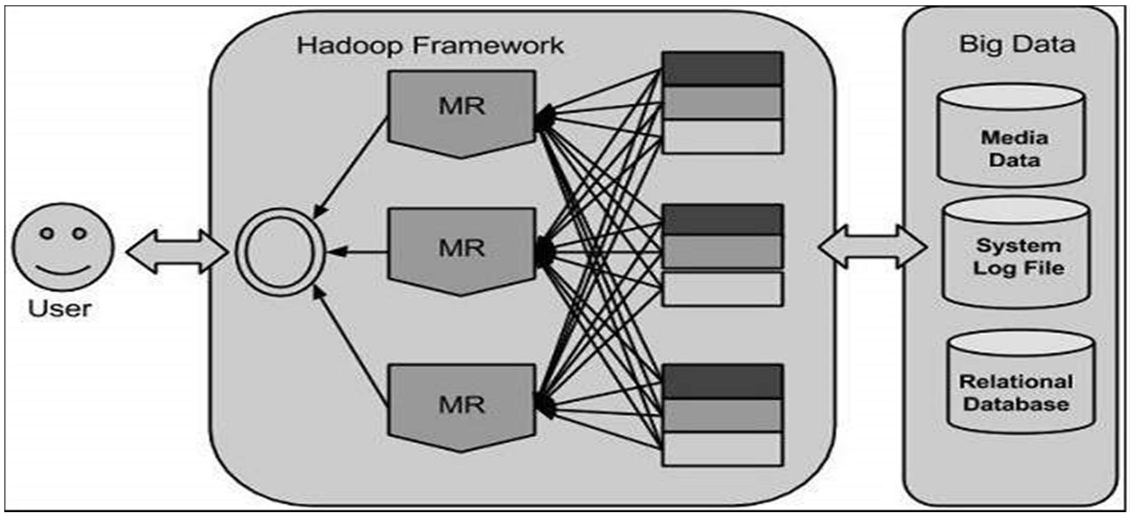
BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJASTHAN)

Topic: Apache Hadoop

1. Introduction

1.1. What is Hadoop?

Hadoop is an open-source software framework for storing and processing data on commodity hardware clusters. It has a lot of storage for any kind of data, a lot of processing power, and it can handle almost unlimited concurrent processes or jobs.
Hadoop runs applications using the MapReduce algorithm, where the data is processed in parallel with others. In short, Hadoop is used to develop applications that could perform complete statistical analysis on huge amounts of data**.**



1.2. Why is Hadoop important?

- **Flexibility:** Unlike traditional relational databases, you don’t have to preprocess data before storing it. You can store as much data as you want and decide how to use it later. That includes unstructured data like text, images and videos.
- **Computing power:** Hadoop's distributed computing model processes big data fast. The more computing nodes you use the more processing power you have.
- **Scalability:** You can easily grow your system to handle more data simply by adding nodes. Little administration is required.

- **Low cost**: The open-source framework is free and uses commodity hardware to store large quantities of data.
- **Fault tolerance:** Data and application processing are protected against hardware failure. If a node goes down, jobs are automatically redirected to other nodes to make sure the distributed computing does not fail. Multiple copies of all data are stored automatically.

1.3. How Hadoop Works?

Currently, four core modules are included in the basic framework from the Apache Foundation:

- **Hadoop Common** – the libraries and utilities used by other Hadoop modules.
- **Hadoop Distributed File System (HDFS)** – the Java-based scalable system that stores data across multiple machines without prior organization.
- **YARN** – (Yet Another Resource Negotiator) provides resource management for the processes running on Hadoop.
- **MapReduce** – a parallel processing software framework. It consists of two steps. Map step is a master node that takes inputs and partitions them into smaller sub problems and then distributes them to worker nodes. After the map step has taken place, the master node takes the answers to all of the sub problems and combines them to produce output.

2. Hadoop Architecture

At its core, Hadoop has two major layers namely −

- Processing/Computation layer (MapReduce)
- Storage layer (Hadoop Distributed File System)

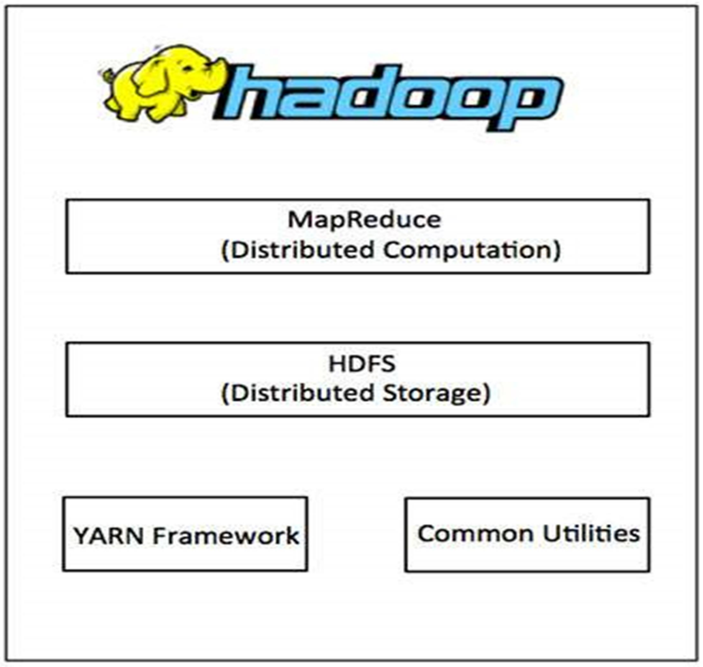
MapReduce: MapReduce is a parallel programming model for writing distributed applications devised at Google for efficient processing of large amounts of data (multi-terabyte data-sets), on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. The MapReduce program runs on Hadoop which is an Apache open-source framework.

Hadoop Distributed File System:

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. It is highly fault-tolerant and is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications having large datasets.

Apart from the above-mentioned two core components, Hadoop framework also includes the following two modules −

- Hadoop Common −These are Java libraries and utilities required by other Hadoop modules.
- Hadoop YARN −This is a framework for job scheduling and cluster resource management.



2.1. HDFC Architecture

The architecture of HDFS is shown in the below figure.

For an HDFS service, we have a NameNode that has the master process running on one of the machines and DataNodes, which are the slave nodes.

NameNode: NameNode is the master service that hosts metadata in disk and RAM. It holds information about the various DataNodes, their location, the size of each block, etc.

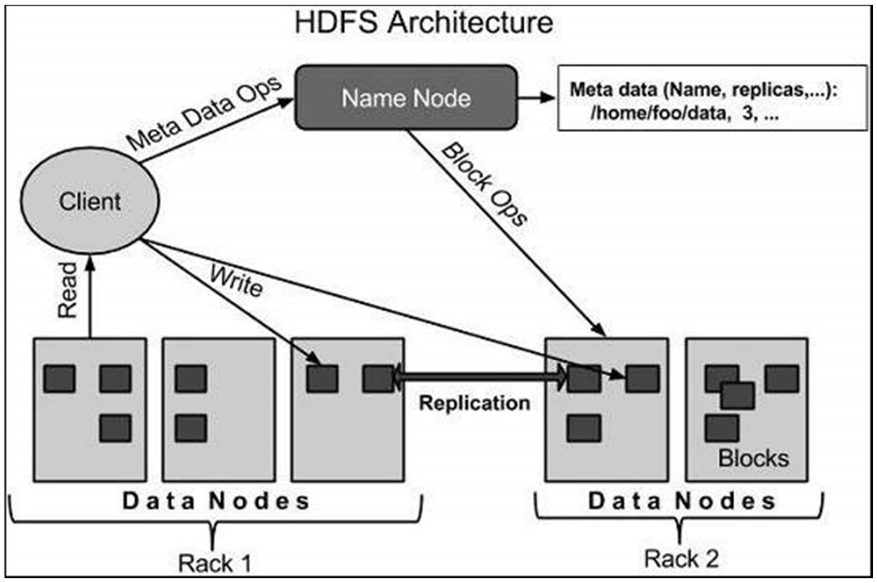


Fig: HDFS Architecture

DataNode: DataNodes hold the actual data blocks and send block reports to the NameNode every 10 seconds. The DataNode stores and retrieves the blocks when the NameNode asks. It reads and writes the client's request and performs block creation, deletion, and replication based on instructions from the NameNode.

- Data that is written to HDFS is split into blocks, depending on its size. The blocks are randomly distributed across the nodes. With the auto-replication feature, these blocks are auto-replicated across multiple machines with the condition that no two identical blocks can sit on the same machine.
- As soon as the cluster comes up, the DataNodes start sending their heartbeats to the NameNodes every three seconds. The NameNode stores this information; in other words, it starts building metadata in RAM, which contains information about the DataNodes available in the beginning. This metadata is maintained in RAM, as well as in the disk.

3. Installation Steps

>>> Before installing hadoop, update the system by following command:

```
sudo apt update
```

Before installing Hadoop into the Linux environment, we need to set up Linux using ssh (Secure Shell). Follow the steps given below for setting up the Linux environment.

Creating a User:

It is recommended to create a separate user for Hadoop to isolate the Hadoop file system from the Unix file system. Follow the steps given below to create a user:

- **Step 1:** Create a user from the root account using the command "*sudo adduser username*".
- **Step 2:** Now you can open an existing user account using the command "*su - username*".

>>> Open the Linux terminal and type the following commands to create a user.

```
sudo adduser hdoop
  password:
```

***Note: hdoop** is a user given name.*

>>> Add **hdoop** as sudo user

```
sudo adduser hdoop sudo
```

Installing Java:

Java is the main prerequisite for Hadoop. First of all, you should verify the existence of java in your system using the command:

```
java -version
```

If java is not installed in your system, then follow the steps given below for installing java.

Step 1: Downloading java.

>>> Go to the below link to download java.
https://download.oracle.com/java/17/latest/jdk-17_linux-x64_bin.tar.gz

>>> After the download is complete, extract the files and go inside the directory "jdk-17_linux-x64_bin"
To make java available to all the users, you have to move it to the location "/usr/local/".
Type the following commands:

```
sudo mv jdk-17.0.4.1 /usr/local/
```

Step 2: Setting environment path in bashrc file

>>> Open bashrc file using:

```
sudo nano .bashrc
```

>>> paste these lines inside the file:

```
export JAVA_HOME=/usr/local/jdk-17.0.4.1
export PATH=$PATH:$JAVA_HOME/bin
```

Note**: Now, press **ctrl+s** to **save** the file and **ctrl+x** to **exit

>>> Now, run the following command to apply the changes in the running system:

```
source  ~/.bashrc
```

Step 3: Use the following commands to configure java alternatives −

```
$ sudo update-alternatives --install /usr/bin/java java /usr/local/jdk-17.0.4.1/bin/java 1
$ sudo update-alternatives --install /usr/bin/javac javac /usr/local/ jdk-17.0.4.1/bin/javac 1
$ sudo update-alternatives --install /usr/bin/jar jar /usr/local/jdk-17.0.4.1/bin/jar 1
$ sudo update-alternatives --set java /usr/local/jdk-17.0.4.1/bin/java
$ sudo update-alternatives --set javac /usr/local/jdk-17.0.4.1/bin/javac
$ sudo update-alternatives --set jar /usr/local/jdk-17.0.4.1/bin/jar
```

Now, java is successfully installed in your system. You can verify it using the command "*java -version*"

>>> Switch to the newly created user

```
su - hdoop
```

SSH Setup and Key Generation:

SSH setup is required to do different operations on a cluster such as starting, stopping, distributed daemon shell operations. To authenticate different users of Hadoop, it is required to provide a public/private key pair for a Hadoop user and share it with different users.

```
sudo apt install openssh-server openssh-client -y
```

The following commands are used for generating a key value pair using SSH. Copy the public keys from id_rsa.pub to authorized_keys, and provide the owner with read and write permissions to authorized_keys file respectively

```
$ ssh-keygen -t rsa –P ‘ ‘ -f ~/.ssh/id_rsa

$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

$ chmod 0600 ~/.ssh/authorized_keys
```

Downloading Hadoop

>>> Download and extract Hadoop 3.3.4 from Apache software foundation using the following commands.

```
$ wget https://downloads.apache.org/hadoop/common/hadoop-3.3.4/hadoop-3.3.4.tar.gz

$ tar xzf hadoop-3.3.4.tar.gz

$ exit
```

Hadoop Operation Modes:

Once you have downloaded Hadoop, you can operate your Hadoop cluster in one of the three supported modes −
Local/Standalone Mode −After downloading Hadoop in your system, by default, it is configured in a standalone mode and can be run as a single java process.

Pseudo Distributed Mode − It is a distributed simulation on a single machine. Each Hadoop daemon such as hdfs, yarn, MapReduce etc., will run as a separate java process. This mode is useful for development.

Fully Distributed Mode −This mode is fully distributed with a minimum of two or more machines as a cluster.

Installing Hadoop in Standalone Mode:

Here we will discuss the installation of Hadoop 3.3.4 in standalone mode. There are no daemons running and everything runs in a single JVM. Standalone mode is suitable for running MapReduce programs during development, since it is easy to test and debug them.

Setting Up Hadoop:

You can set Hadoop environment variables by appending the following commands to *~/.bashrc* file.

```
sudo nano .bashrc
```

#Add below lines in this file

```
export HADOOP_HOME=/home/hdoop/hadoop-3.3.4
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
```

Save the file using *ctrl+s* and exit using c*trl+x*

>>> Apply the changes using the command:

```
source ~/.bashrc
```

>>> Before proceeding further, you need to make sure that Hadoop is working fine. Just issue the following command –

```
$ hadoop version
```

If everything is fine with your setup, then you should see the following result –

```
Hadoop 3.3.4
Subversion https://svn.apache.org/repos/asf/hadoop/common -r 1529768
Compiled by hortonmu on 2013-10-07T06:28Z
Compiled with protoc 2.5.0

From source with checksum 79e53ce7994d1628b240f09af91e1af4
```

It means your Hadoop's standalone mode setup is working fine. By default, Hadoop is configured to run in a non-distributed mode on a single machine.

Examples

Example1. Word count Example using Map-Reduce:

Hadoop installation delivers the example MapReduce jar file, which provides basic functionality of MapReduce and can be used for calculating word counts in a given list of files.

Let's have an input directory where we will push a few files and our requirement is to count the total number of words in those files. To calculate the total number of words, we do not need to write our MapReduce, provided the .jar file contains the implementation for word count. You can try other examples using the same .jar file; just issue the following commands to check supported MapReduce functional programs by hadoop-mapreduce-examples-3.3.4.jar file.

Step 1: Create temporary content files in the input directory. You can create this input directory anywhere you would like to work.

```
$ mkdir input
$ cp $HADOOP_HOME/*.txt input
$ ls -l input
```

It will give the following files in your input directory −

```
total 24
-rw-r--r-- 1 root root 15164 Feb 21 10:14 LICENSE.txt
-rw-r--r-- 1 root root   101 Feb 21 10:14 NOTICE.txt
-rw-r--r-- 1 root root  1366 Feb 21 10:14 README.txt
```

These files have been copied from the Hadoop installation home directory. For your experiment, you can have different and large sets of files.

Step 2: Let's start the Hadoop process to count the total number of words in all the files available in the input directory, as follows −

```
$ hadoop  jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-
examples-3.3.4.jar  wordcount input output
```

Step 3: Step-2 will do the required processing and save the output in output/part-r-00000 file, which you can check by using −

```
$ cat output/*
```

It will list down all the words along with their total counts available in all the files available in the input directory.

```
 "AS “     4

"Contribution" 1
"Contributor" 1
"Derivative 1
"Legal 1
"License"      1
"License");     1
"Licensor"      1
"NOTICE”       1
"Not      1
"Object"        1
"Source”        1
"Work”     1
"You"     1
"Your")   1
"control"       1
```

Example2. Inverted Index Example using Map-Reduce:

The inverted index is a database index storing a mapping from content, such as words or numbers, to its locations in a database, or in a document or a set of documents. The purpose of an inverted index is to allow a fast full-text search.

To submit a Hadoop job, the MapReduce implementation should be packaged as a **jar** file. In this example, Hadoop job gets two text files from the "**input**" folder as the arguments of the Mapper.

```
#file01
5722018411  Hello World Bye World
#file02
6722018415  Hello Hadoop Goodbye Hadoop
```

And by submitting a Hadoop job and applying Reduce step, it generates an inverted index as below:

Word	**File : Frequency**
bye	5722018411:1
goodbye	6722018415:1
hadoop	6722018415:2
hello	5722018411:1 6722018415:1
world	5722018411:2

>>> Create a directory using mkdir command:

```
mkdir example2
```

>>> Move the InvertedIndex.java and input folder to the example2 directory

Note: *InvertedIndex.java and input folder are provided with lab sheet.*

Run the following commands to compile InvertedIndex.java and create a **jar** file.

>>> To compile InvertedIndex.java

```
$ /home/hdoop/hadoop-3.3.4/bin/hadoop com.sun.tools.javac.Main InvertedIndex.java
```

>>> To create a jar file

```
$ jar cf invertedindex.jar InvertedIndex*.class
```

>>> Run the following command to submit the job, get the input files from input folder, generate the inverted index and store its output in the output folder:

Syntax: hadoop jar <path of jar file> <class _name> <path of input directory> <path where you want to create output directory>

```
$ hadoop jar /home/hdoop/example2/invertedindex.jar InvertedIndex
/home/hdoop/example2/input /home/hdoop/example2/output
```

>>> And finally to see the output, run the below command:

```
$ cat example2/output/*
```

***********************Word Count************************

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Questions:

1. What are the different Hadoop configuration files?
2. What are the differences between regular FileSystem and HDFS?
3. If you have an input file of 350 MB, how many input splits would HDFS create and what would be the size of each input split?
4. Explain the Storage Unit In Hadoop (HDFS).
5. Compare the main differences between HDFS (Hadoop Distributed File System) and Network Attached Storage (NAS)?
6. Name some of the essential Hadoop tools for effective working with Big Data.