HADOOP - QUICK GUIDE

HADOOP - BIG DATA OVERVIEW

“90% of the world’s data was generated in the last few years.”

Due to the advent of new technologies, devices, and communication means like social networking sites, the amount of data produced by mankind is growing rapidly every year. The amount of data produced by us from the beginning of time till 2003 was 5 billion gigabytes. If you pile up the data in the form of disks it may fill an entire football field. The same amount was created in every two days in 2011, and in every ten minutes in 2013. This rate is still growing enormously. Though all this information produced is meaningful and can be useful when processed, it is being neglected.

What is Big Data?

Big Data is a collection of large datasets that cannot be processed using traditional computing techniques. It is not a single technique or a tool, rather it involves many areas of business and technology.

What Comes Under Big Data?

Big data involves the data produced by different devices and applications. Given below are some of the fields that come under the umbrella of Big Data.

- **Black Box Data** : It is a component of helicopter, airplanes, and jets, etc. It captures voices of the flight crew, recordings of microphones and earphones, and the performance information of the aircraft.
- **Social Media Data** : Social media such as Facebook and Twitter hold information and the views posted by millions of people across the globe.
- **Stock Exchange Data** : The stock exchange data holds information about the ‘buy’ and ‘sell’ decisions made on a share of different companies made by the customers.
- **Power Grid Data** : The power grid data holds information consumed by a particular node with respect to a base station.
- **Transport Data** : Transport data includes model, capacity, distance and availability of a vehicle.
- **Search Engine Data** : Search engines retrieve lots of data from different databases.

Thus Big Data includes huge volume, high velocity, and extensible variety of data. The data in it will be of three types.

- **Structured data** : Relational data.
- **Semi Structured data** : XML data.
- **Unstructured data** : Word, PDF, Text, Media Logs.

Benefits of Big Data

- Using the information kept in the social network like Facebook, the marketing agencies are learning about the response for their campaigns, promotions, and other advertising mediums.
- Using the information in the social media like preferences and product perception of their consumers, product companies and retail organizations are planning their production.
- Using the data regarding the previous medical history of patients, hospitals are providing better and quick service.

Big Data Technologies

Big data technologies are important in providing more accurate analysis, which may lead to more concrete decision-making resulting in greater operational efficiencies, cost reductions, and reduced risks for the business.

To harness the power of big data, you would require an infrastructure that can manage and process huge volumes of structured and unstructured data in realtime and can protect data privacy and security.

There are various technologies in the market from different vendors including Amazon, IBM, Microsoft, etc., to handle big data. While looking into the technologies that handle big data, we examine the following two classes of technology:

Operational Big Data

These include systems like MongoDB that provide operational capabilities for real-time, interactive workloads where data is primarily captured and stored.

NoSQL Big Data systems are designed to take advantage of new cloud computing architectures that have emerged over the past decade to allow massive computations to be run inexpensively and efficiently. This makes operational big data workloads much easier to manage, cheaper, and faster to implement.

Some NoSQL systems can provide insights into patterns and trends based on real-time data with minimal coding and without the need for data scientists and additional infrastructure.

Analytical Big Data

These includes systems like Massively Parallel Processing MPP database systems and MapReduce that provide analytical capabilities for retrospective and complex analysis that may touch most or all of the data.

MapReduce provides a new method of analyzing data that is complementary to the capabilities provided by SQL, and a system based on MapReduce that can be scaled up from single servers to thousands of high and low end machines.

These two classes of technology are complementary and frequently deployed together.

Operational vs. Analytical Systems

	Operational	Analytical

Latency	1 ms - 100 ms	1 min - 100 min
Concurrency	1000 - 100,000	1 - 10
Access Pattern	Writes and Reads	Reads
Queries	Selective	Unselective
Data Scope	Operational	Retrospective
End User	Customer	Data Scientist
Technology	NoSQL	MapReduce, MPP Database

Big Data Challenges

The major challenges associated with big data are as follows:

- Capturing data
- Curation
- Storage
- Searching
- Sharing
- Transfer
- Analysis
- Presentation

To fulfill the above challenges, organizations normally take the help of enterprise servers.

HADOOP - BIG DATA SOLUTIONS

Traditional Enterprise Approach

In this approach, an enterprise will have a computer to store and process big data. For storage purpose, the programmers will take the help of their choice of database vendors such as Oracle, IBM, etc. In this approach, the user interacts with the application, which in turn handles the part of data storage and analysis.

Limitation

This approach works fine with those applications that process less voluminous data that can be accommodated by standard database servers, or up to the limit of the processor that is processing the data. But when it comes to dealing with huge amounts of scalable data, it is a hectic task to process such data through a single database bottleneck.

Google’s Solution

Google solved this problem using an algorithm called MapReduce. This algorithm divides the task into small parts and assigns them to many computers, and collects the results from them which when integrated, form the result dataset.

Hadoop

Using the solution provided by Google, Doug Cutting and his team developed an Open Source Project called HADOOP.

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.

HADOOP - INTRODUCTION TO HADOOP

Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. The Hadoop framework application works in an environment that provides distributed storage and computation across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

Hadoop Architecture

At its core, Hadoop has two major layers namely:

- Processing/Computation layer *MapReduce*, and
- Storage layer *HadoopDistributedFileSystem*.

MapReduce

MapReduce is a parallel programming model for writing distributed applications devised at Google for efficient processing of large amounts of data *multiterabytedata − sets*, on large clusters *thousandsofnodes* 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.

How Does Hadoop Work?

It is quite expensive to build bigger servers with heavy configurations that handle large scale processing, but as an alternative, you can tie together many commodity computers with single-CPU, as a single functional distributed system and practically, the clustered machines can read the dataset in parallel and provide a much higher throughput. Moreover, it is cheaper than one high-end server. So this is the first motivational factor behind using Hadoop that it runs across clustered and low-cost machines.

Hadoop runs code across a cluster of computers. This process includes the following core tasks that Hadoop performs:

- Data is initially divided into directories and files. Files are divided into uniform sized blocks of 128M and 64M *preferably*128*M*.
- These files are then distributed across various cluster nodes for further processing.
- HDFS, being on top of the local file system, supervises the processing.
- Blocks are replicated for handling hardware failure.

- Checking that the code was executed successfully.
- Performing the sort that takes place between the map and reduce stages.
- Sending the sorted data to a certain computer.
- Writing the debugging logs for each job.

Advantages of Hadoop

- Hadoop framework allows the user to quickly write and test distributed systems. It is efficient, and it automatic distributes the data and work across the machines and in turn, utilizes the underlying parallelism of the CPU cores.
- Hadoop does not rely on hardware to provide fault-tolerance and high availability *FTHA*, rather Hadoop library itself has been designed to detect and handle failures at the application layer.
- Servers can be added or removed from the cluster dynamically and Hadoop continues to operate without interruption.
- Another big advantage of Hadoop is that apart from being open source, it is compatible on all the platforms since it is Java based.

HADOOP - ENVIORNMENT SETUP

Hadoop is supported by GNU/Linux platform and its flavors. Therefore, we have to install a Linux operating system for setting up Hadoop environment. In case you have an OS other than Linux, you can install a Virtualbox software in it and have Linux inside the Virtualbox.

Pre-installation Setup

Before installing Hadoop into the Linux environment, we need to set up Linux using ssh *SecureShell*. Follow the steps given below for setting up the Linux environment.

Creating a User

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

- Open the root using the command “su”.
- Create a user from the root account using the command “useradd username”.
- 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.

```
$ su
   password:
# useradd hadoop
# passwd hadoop
   New passwd:
   Retype new passwd
```

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 public/private key pair for a Hadoop user and share it with different users.

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

```
$ ssh-keygen -t rsa
```

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 0600 ~/.ssh/authorized_keys
```

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”. The syntax of java version command is given below.

```
$ java -version
```

If everything is in order, it will give you the following output.

```
java version "1.7.0_71"
Java(TM) SE Runtime Environment (build 1.7.0_71-b13)
Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode)
```

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

Step 1

Download java $JDK < latestversion > - X64.tar.gz$ by visiting the following link http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads1880260.html.

Then jdk-7u71-linux-x64.tar.gz will be downloaded into your system.

Step 2

Generally you will find the downloaded java file in Downloads folder. Verify it and extract the jdk-7u71-linux-x64.gz file using the following commands.

```
$ cd Downloads/
$ ls
jdk-7u71-linux-x64.gz
$ tar zxf jdk-7u71-linux-x64.gz
$ ls
jdk1.7.0_71   jdk-7u71-linux-x64.gz
```

Step 3

To make java available to all the users, you have to move it to the location “/usr/local/”. Open root, and type the following commands.

```
$ su
password:
# mv jdk1.7.0_71 /usr/local/
# exit
```

Step 4

For setting up PATH and JAVA_HOME variables, add the following commands to ~/.bashrc file.

```
export JAVA_HOME=/usr/local/jdk1.7.0_71
export PATH=PATH:$JAVA_HOME/bin
```

Now verify the java -version command from the terminal as explained above.

Downloading Hadoop

Download and extract Hadoop 2.4.1 from Apache software foundation using the following commands.

```
$ su
```

```
password:
# cd /usr/local
# wget http://apache.claz.org/hadoop/common/hadoop-2.4.1/
hadoop-2.4.1.tar.gz
# tar xzf hadoop-2.4.1.tar.gz
# mv hadoop-2.4.1/* to hadoop/
# 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 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 minimum two or more machines as a cluster. We will come across this mode in detail in the coming chapters.

Installing Hadoop in Standalone Mode

Here we will discuss the installation of **Hadoop 2.4.1** 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.

```
export HADOOP_HOME=/usr/local/hadoop
```

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 2.4.1
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.

Example

Let's check a simple example of Hadoop. Hadoop installation delivers the following example MapReduce jar file, which provides basic functionality of MapReduce and can be used for calculating, like Pi value, word counts in a given list of files, etc.

```
$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar
```

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-2.2.0.jar file.

```
$ hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduceexamples-2.2.0.jar
```

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-mapreduceexamples-2.2.0.jar
wordcount input ouput
```

Step 3

Step-2 will do the required processing and save the output in output/part-r00000 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
"[]"     1
"control"       1
"printed        1
"submitted"     1
(50%)    1
(BIS),   1
```

```
(C)       1
(Don't)   1
(ECCN)    1
(INCLUDING      2
(INCLUDING,     2
.............
```

Installing Hadoop in Pseudo Distributed Mode

Follow the steps given below to install Hadoop 2.4.1 in pseudo distributed mode.

Step 1: Setting Up Hadoop

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

```
export HADOOP_HOME=/usr/local/hadoop
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_INSTALL=$HADOOP_HOME
```

Now apply all the changes into the current running system.

```
$ source ~/.bashrc
```

Step 2: Hadoop Configuration

You can find all the Hadoop configuration files in the location “$HADOOP_HOME/etc/hadoop”. It is required to make changes in those configuration files according to your Hadoop infrastructure.

```
$ cd $HADOOP_HOME/etc/hadoop
```

In order to develop Hadoop programs in java, you have to reset the java environment variables in **hadoop-env.sh** file by replacing **JAVA_HOME** value with the location of java in your system.

```
export JAVA_HOME=/usr/local/jdk1.7.0_71
```

The following are the list of files that you have to edit to configure Hadoop.

core-site.xml

The **core-site.xml** file contains information such as the port number used for Hadoop instance, memory allocated for the file system, memory limit for storing the data, and size of Read/Write buffers.

Open the core-site.xml and add the following properties in between <configuration>, </configuration> tags.

```
<configuration>

   <property>
      <name>fs.default.name </name>
      <value> hdfs://localhost:9000 </value>
   </property>

</configuration>
```

hdfs-site.xml

The **hdfs-site.xml** file contains information such as the value of replication data, namenode path,

and datanode paths of your local file systems. It means the place where you want to store the Hadoop infrastructure.

Let us assume the following data.

```
dfs.replication (data replication value) = 1
(In the below given path /hadoop/ is the user name.
hadoopinfra/hdfs/namenode is the directory created by hdfs file system.)
namenode path = //home/hadoop/hadoopinfra/hdfs/namenode
(hadoopinfra/hdfs/datanode is the directory created by hdfs file system.)
datanode path = //home/hadoop/hadoopinfra/hdfs/datanode
```

Open this file and add the following properties in between the <configuration> </configuration> tags in this file.

```
<configuration>

   <property>
      <name>dfs.replication</name>
      <value>1</value>
   </property>

   <property>
      <name>dfs.name.dir</name>
      <value>file:///home/hadoop/hadoopinfra/hdfs/namenode </value>
   </property>

   <property>
      <name>dfs.data.dir</name>
      <value>file:///home/hadoop/hadoopinfra/hdfs/datanode </value>
   </property>

</configuration>
```

Note: In the above file, all the property values are user-defined and you can make changes according to your Hadoop infrastructure.

yarn-site.xml

This file is used to configure yarn into Hadoop. Open the yarn-site.xml file and add the following properties in between the <configuration>, </configuration> tags in this file.

```
<configuration>

   <property>
      <name>yarn.nodemanager.aux-services</name>
      <value>mapreduce_shuffle</value>
   </property>

</configuration>
```

mapred-site.xml

This file is used to specify which MapReduce framework we are using. By default, Hadoop contains a template of yarn-site.xml. First of all, it is required to copy the file from **mapred-site,xml.template** to **mapred-site.xml** file using the following command.

```
$ cp mapred-site.xml.template mapred-site.xml
```

Open mapred-site.xml file and add the following properties in between the <configuration>, </configuration>tags in this file.

```
<configuration>

   <property>
      <name>mapreduce.framework.name</name>
```

```
      <value>yarn</value>
   </property>

</configuration>
```

Verifying Hadoop Installation

The following steps are used to verify the Hadoop installation.

Step 1: Name Node Setup

Set up the namenode using the command “hdfs namenode -format” as follows.

```
$ cd ~
$ hdfs namenode -format
```

The expected result is as follows.

```
10/24/14 21:30:55 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = localhost/192.168.1.11
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 2.4.1
...
...
10/24/14 21:30:56 INFO common.Storage: Storage directory
/home/hadoop/hadoopinfra/hdfs/namenode has been successfully formatted.
10/24/14 21:30:56 INFO namenode.NNStorageRetentionManager: Going to
retain 1 images with txid >= 0
10/24/14 21:30:56 INFO util.ExitUtil: Exiting with status 0
10/24/14 21:30:56 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at localhost/192.168.1.11
************************************************************/
```

Step 2: Verifying Hadoop dfs

The following command is used to start dfs. Executing this command will start your Hadoop file system.

```
$ start-dfs.sh
```

The expected output is as follows:

```
10/24/14 21:37:56
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/hadoop/hadoop
2.4.1/logs/hadoop-hadoop-namenode-localhost.out
localhost: starting datanode, logging to /home/hadoop/hadoop
2.4.1/logs/hadoop-hadoop-datanode-localhost.out
Starting secondary namenodes [0.0.0.0]
```

Step 3: Verifying Yarn Script

The following command is used to start the yarn script. Executing this command will start your yarn daemons.

```
$ start-yarn.sh
```

The expected output as follows:

```
starting yarn daemons
```

```
starting resourcemanager, logging to /home/hadoop/hadoop
2.4.1/logs/yarn-hadoop-resourcemanager-localhost.out
localhost: starting nodemanager, logging to /home/hadoop/hadoop
2.4.1/logs/yarn-hadoop-nodemanager-localhost.out
```

Step 4: Accessing Hadoop on Browser

The default port number to access Hadoop is 50070. Use the following url to get Hadoop services on browser.

```
http://localhost:50070/
```

Step 5: Verify All Applications for Cluster

The default port number to access all applications of cluster is 8088. Use the following url to visit this service.

```
http://localhost:8088/
```

HADOOP - HDFS OVERVIEW

Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly faulttolerant and designed using low-cost hardware.

HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

Features of HDFS

- It is suitable for the distributed storage and processing.
- Hadoop provides a command interface to interact with HDFS.
- The built-in servers of namenode and datanode help users to easily check the status of cluster.
- Streaming access to file system data.
- HDFS provides file permissions and authentication.

HDFS Architecture

Given below is the architecture of a Hadoop File System.

HDFS follows the master-slave architecture and it has the following elements.

Namenode

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks:

- Manages the file system namespace.
- Regulates client's access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories.

Datanode

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node *Commodityhardware/System* in a cluster, there will be a datanode. These nodes manage the data storage of their system.

- Datanodes perform read-write operations on the file systems, as per client request.
- They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

Block

Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

Goals of HDFS

- **Fault detection and recovery** : Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.
- **Huge datasets** : HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.
- **Hardware at data** : A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

HADOOP - HDFS OPERATIONS

Starting HDFS

Initially you have to format the configured HDFS file system, open namenode *HDFSserver*, and execute the following command.

```
$ hadoop namenode -format
```

After formatting the HDFS, start the distributed file system. The following command will start the namenode as well as the data nodes as cluster.

```
$ start-dfs.sh
```

Listing Files in HDFS

After loading the information in the server, we can find the list of files in a directory, status of a file, using ‘ls’. Given below is the syntax of ls that you can pass to a directory or a filename as an argument.

```
$ $HADOOP_HOME/bin/hadoop fs -ls <args>
```

Inserting Data into HDFS

Assume we have data in the file called file.txt in the local system which is ought to be saved in the hdfs file system. Follow the steps given below to insert the required file in the Hadoop file system.

Step 1

You have to create an input directory.

```
$ $HADOOP_HOME/bin/hadoop fs -mkdir /user/input
```

Step 2

Transfer and store a data file from local systems to the Hadoop file system using the put command.

```
$ $HADOOP_HOME/bin/hadoop fs -put /home/file.txt /user/input
```

Step 3

You can verify the file using ls command.

```
$ $HADOOP_HOME/bin/hadoop fs -ls /user/input
```

Retrieving Data from HDFS

Assume we have a file in HDFS called outfile. Given below is a simple demonstration for retrieving the required file from the Hadoop file system.

Step 1

Initially, view the data from HDFS using cat command.

```
$ $HADOOP_HOME/bin/hadoop fs -cat /user/output/outfile
```

Step 2

Get the file from HDFS to the local file system using get command.

```
$ $HADOOP_HOME/bin/hadoop fs -get /user/output/ /home/hadoop_tp/
```

Shutting Down the HDFS

You can shut down the HDFS by using the following command.

```
$ stop-dfs.sh
```

HADOOP - COMMAND REFERENCE

HDFS Command Reference

There are many more commands in **"$HADOOP_HOME/bin/hadoop fs"** than are demonstrated here, although these basic operations will get you started. Running ./bin/hadoop dfs with no additional arguments will list all the commands that can be run with the FsShell system. Furthermore, **$HADOOP_HOME/bin/hadoop fs -help** commandName will display a short usage summary for the operation in question, if you are stuck.

A table of all the operations is shown below. The following conventions are used for parameters:

```
"<path>" means any file or directory name.
"<path>..." means one or more file or directory names.
"<file>" means any filename.
"<src>" and "<dest>" are path names in a directed operation.
"<localSrc>" and "<localDest>" are paths as above, but on the local file system.
```

All other files and path names refer to the objects inside HDFS.

Command	Description
-ls <path>	Lists the contents of the directory specified by path, showing the names, permissions, owner, size and modification date for each entry.
-lsr <path>	Behaves like -ls, but recursively displays entries in all subdirectories of path.
-du <path>	Shows disk usage, in bytes, for all the files which match path; filenames are reported with the full HDFS protocol prefix.
-dus <path>	Like -du, but prints a summary of disk usage of all files/directories in the path.
-mv <src><dest>	Moves the file or directory indicated by src to dest, within HDFS.
-cp <src> <dest>	Copies the file or directory identified by src to dest, within HDFS.
-rm <path>	Removes the file or empty directory identified by path.
-rmr <path>	Removes the file or directory identified by path. Recursively deletes any child entries $i.e.,filesorsubdirectoriesofpath$.
-put <localSrc> <dest>	Copies the file or directory from the local file system identified by localSrc to dest within the DFS.
-copyFromLocal <localSrc> <dest>	Identical to -put
-moveFromLocal <localSrc> <dest>	Copies the file or directory from the local file system identified by localSrc to dest within HDFS, and then deletes the local copy on success.
-get [-crc] <src> <localDest>	Copies the file or directory in HDFS identified by src to the local file system path identified by localDest.
-getmerge <src> <localDest>	Retrieves all files that match the path src in HDFS, and copies them to a single, merged file in the local file system identified by localDest.
-cat <filen-ame>	Displays the contents of filename on stdout.
-copyToLocal <src> <localDest>	Identical to -get
-moveToLocal <src> <localDest>	Works like -get, but deletes the HDFS copy on success.
-mkdir <path>	Creates a directory named path in HDFS. Creates any parent directories in path that are missing $e.g., mkdir-pinLinux$.
-setrep [-R] [-w] rep <path>	Sets the target replication factor for files identified by path to rep. $Theactualreplicationfactorwillmovetowardthetargetovertime$
-touchz <path>	Creates a file at path containing the current time as a timestamp. Fails if a file already exists at path, unless the file is already size 0.
-test -[ezd] <path>	Returns 1 if path exists; has zero length; or is a directory or 0 otherwise.

-stat [format] <path>	Prints information about path. Format is a string which accepts file size in blocks , filename , block size , replication , and modification date .
-tail [-f] <file2name>	Shows the last 1KB of file on stdout.
-chmod [-R] mode,mode,... <path>...	Changes the file permissions associated with one or more objects identified by path.... Performs changes recursively with R. mode is a 3-digit octal mode, or {augo}+/-{rwxX}. Assumes if no scope is specified and does not apply an umask.
-chown [-R] [owner][:[group]] <path>...	Sets the owning user and/or group for files or directories identified by path.... Sets owner recursively if -R is specified.
chgrp [-R] group <path>...	Sets the owning group for files or directories identified by path.... Sets group recursively if -R is specified.
-help <cmd-name>	Returns usage information for one of the commands listed above. You must omit the leading '-' character in cmd.

HADOOP - MAPREDUCE

MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner.

What is MapReduce?

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples *key/valuepairs*. Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

The Algorithm

- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
    - **Map stage** : The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in the Hadoop file system *HDFS*. The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.
    - **Reduce stage** : This stage is the combination of the **Shuffle** stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.
- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task

completion, and copying data around the cluster between the nodes.

- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

Inputs and Outputs *JavaPerspective*

The MapReduce framework operates on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.

The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface. Additionally, the key classes have to implement the Writable-Comparable interface to facilitate sorting by the framework. Input and Output types of a MapReduce job: *Input* <k1, v1> -> map -> <k2, v2>-> reduce -> <k3, v3>*Output*.

	Input	Output
Map	<k1, v1>	list $< k2, v2 >$
Reduce	<k2, list$v2$>	list $< k3, v3 >$

Terminology

- **PayLoad** - Applications implement the Map and the Reduce functions, and form the core of the job.
- **Mapper** - Mapper maps the input key/value pairs to a set of intermediate key/value pair.
- **NamedNode** - Node that manages the Hadoop Distributed File System *HDFS*.
- **DataNode** - Node where data is presented in advance before any processing takes place.
- **MasterNode** - Node where JobTracker runs and which accepts job requests from clients.
- **SlaveNode** - Node where Map and Reduce program runs.
- **JobTracker** - Schedules jobs and tracks the assign jobs to Task tracker.
- **Task Tracker** - Tracks the task and reports status to JobTracker.
- **Job** - A program is an execution of a Mapper and Reducer across a dataset.
- **Task** - An execution of a Mapper or a Reducer on a slice of data.
- **Task Attempt** - A particular instance of an attempt to execute a task on a SlaveNode.

Example Scenario

Given below is the data regarding the electrical consumption of an organization. It contains the monthly electrical consumption and the annual average for various years.

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Avg
1979	23	23	2	43	24	25	26	26	26	26	25	26	25
1980	26	27	28	28	28	30	31	31	31	30	30	30	29
1981	31	32	32	32	33	34	35	36	36	34	34	34	34

```
1984  39   38   39   39   39   41   42  43   40   39   38   38   40
1985  38   39   39   39   39   41   41  41   00   40   39   39   45
```

If the above data is given as input, we have to write applications to process it and produce results such as finding the year of maximum usage, year of minimum usage, and so on. This is a walkover for the programmers with finite number of records. They will simply write the logic to produce the required output, and pass the data to the application written.

But, think of the data representing the electrical consumption of all the largescale industries of a particular state, since its formation.

When we write applications to process such bulk data,

- They will take a lot of time to execute.
- There will be a heavy network traffic when we move data from source to network server and so on.

To solve these problems, we have the MapReduce framework.

Input Data

The above data is saved as **sample.txt**and given as input. The input file looks as shown below.

```
1979   23   23   2   43   24   25   26   26   26   26   25   26  25
1980   26   27   28  28   28   30   31   31   31   30   30   30  29
1981   31   32   32  32   33   34   35   36   36   34   34   34  34
1984   39   38   39  39   39   41   42   43   40   39   38   38  40
1985   38   39   39  39   39   41   41   41   00   40   39   39  45
```

Example Program

Given below is the program to the sample data using MapReduce framework.

```
package hadoop;

import java.util.*;
import java.io.IOException;
import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;
public class ProcessUnits
{
   //Mapper class
   public static class E_EMapper extends MapReduceBase implements
   Mapper<LongWritable ,/*Input key Type */
   Text,                /*Input value Type*/
   Text,                /*Output key Type*/
   IntWritable>        /*Output value Type*/
   {  //Map function
      public void map(LongWritable key, Text value,
      OutputCollector<Text, IntWritable> output,
      Reporter reporter) throws IOException
      {
         String line = value.toString();
         String lasttoken = null;
         StringTokenizer s = new StringTokenizer(line,"\t");
         String year = s.nextToken();
         while(s.hasMoreTokens()){lasttoken=s.nextToken();}
         int avgprice = Integer.parseInt(lasttoken);
         output.collect(new Text(year), new IntWritable(avgprice));
```

```
      }
   }
   //Reducer class
   public static class E_EReduce extends MapReduceBase implements
   Reducer< Text, IntWritable, Text, IntWritable >
   {  //Reduce function
      public void reduce( Text key, Iterator <IntWritable> values,
         OutputCollector<Text, IntWritable> output, Reporter reporter)  throws
IOException
         {
            int maxavg=30;
            int val=Integer.MIN_VALUE;
            while (values.hasNext())
            {
               if((val=values.next().get())>maxavg)
               {
                  output.collect(key, new IntWritable(val));
               }
            }

         }
   }

   //Main function
   public static void main(String args[])throws Exception
   {
      JobConf conf = new JobConf(Eleunits.class);
      conf.setJobName("max_eletricityunits");
      conf.setOutputKeyClass(Text.class);
      conf.setOutputValueClass(IntWritable.class);
      conf.setMapperClass(E_EMapper.class);
      conf.setCombinerClass(E_EReduce.class);
      conf.setReducerClass(E_EReduce.class);
      conf.setInputFormat(TextInputFormat.class);
      conf.setOutputFormat(TextOutputFormat.class);
      FileInputFormat.setInputPaths(conf, new Path(args[0]));
      FileOutputFormat.setOutputPath(conf, new Path(args[1]));
      JobClient.runJob(conf);
   }
}
```

Save the above program as **ProcessUnits.java.** The compilation and execution of the program is explained below.

Compilation and Execution of Process Units Program

Let us assume we are in the home directory of a Hadoop user *e. g. /home/hadoop*.

Follow the steps given below to compile and execute the above program.

Step 1

The following command is to create a directory to store the compiled java classes.

```
$ mkdir units
```

Step 2

Download **Hadoop-core-1.2.1.jar,** which is used to compile and execute the MapReduce program. Visit the following link http://mvnrepository.com/artifact/org.apache.hadoop/hadoop-core/1.2.1 to download the jar. Let us assume the downloaded folder is **/home/hadoop/.**

Step 3

The following commands are used for compiling the **ProcessUnits.java** program and creating a jar for the program.

```
$ javac -classpath hadoop-core-1.2.1.jar -d units ProcessUnits.java
$ jar -cvf units.jar -C units/ .
```

Step 4

The following command is used to create an input directory in HDFS.

```
$HADOOP_HOME/bin/hadoop fs -mkdir input_dir
```

Step 5

The following command is used to copy the input file named **sample.txt**in the input directory of HDFS.

```
$HADOOP_HOME/bin/hadoop fs -put /home/hadoop/sample.txt input_dir
```

Step 6

The following command is used to verify the files in the input directory.

```
$HADOOP_HOME/bin/hadoop fs -ls input_dir/
```

Step 7

The following command is used to run the Eleunit_max application by taking the input files from the input directory.

```
$HADOOP_HOME/bin/hadoop jar units.jar hadoop.ProcessUnits input_dir output_dir
```

Wait for a while until the file is executed. After execution, as shown below, the output will contain the number of input splits, the number of Map tasks, the number of reducer tasks, etc.

```
INFO mapreduce.Job: Job job_1414748220717_0002
completed successfully
14/10/31 06:02:52
INFO mapreduce.Job: Counters: 49
File System Counters

FILE: Number of bytes read=61
FILE: Number of bytes written=279400
FILE: Number of read operations=0
FILE: Number of large read operations=0
FILE: Number of write operations=0
HDFS: Number of bytes read=546
HDFS: Number of bytes written=40
HDFS: Number of read operations=9
HDFS: Number of large read operations=0
HDFS: Number of write operations=2 Job Counters


   Launched map tasks=2
   Launched reduce tasks=1
   Data-local map tasks=2
   Total time spent by all maps in occupied slots (ms)=146137
   Total time spent by all reduces in occupied slots (ms)=441
   Total time spent by all map tasks (ms)=14613
   Total time spent by all reduce tasks (ms)=44120
   Total vcore-seconds taken by all map tasks=146137

   Total vcore-seconds taken by all reduce tasks=44120
   Total megabyte-seconds taken by all map tasks=149644288
   Total megabyte-seconds taken by all reduce tasks=45178880
```

```
Map-Reduce Framework

Map input records=5
   Map output records=5
   Map output bytes=45
   Map output materialized bytes=67
   Input split bytes=208
   Combine input records=5
   Combine output records=5
   Reduce input groups=5
   Reduce shuffle bytes=6
   Reduce input records=5
   Reduce output records=5
   Spilled Records=10
   Shuffled Maps =2
   Failed Shuffles=0
   Merged Map outputs=2
   GC time elapsed (ms)=948
   CPU time spent (ms)=5160
   Physical memory (bytes) snapshot=47749120
   Virtual memory (bytes) snapshot=2899349504
   Total committed heap usage (bytes)=277684224

File Output Format Counters

   Bytes Written=40
```

Step 8

The following command is used to verify the resultant files in the output folder.

```
$HADOOP_HOME/bin/hadoop fs -ls output_dir/
```

Step 9

The following command is used to see the output in **Part-00000** file. This file is generated by HDFS.

```
$HADOOP_HOME/bin/hadoop fs -cat output_dir/part-00000
```

Below is the output generated by the MapReduce program.

```
1981    34
1984    40
1985    45
```

Step 10

The following command is used to copy the output folder from HDFS to the local file system for analyzing.

```
$HADOOP_HOME/bin/hadoop fs -cat output_dir/part-00000/bin/hadoop dfs get output_dir
/home/hadoop
```

Important Commands

All Hadoop commands are invoked by the **$HADOOP_HOME/bin/hadoop** command. Running the Hadoop script without any arguments prints the description for all commands.

Usage : hadoop [--config confdir] COMMAND

The following table lists the options available and their description.

Options	Description
namenode -format	Formats the DFS filesystem.
secondarynamenode	Runs the DFS secondary namenode.
namenode	Runs the DFS namenode.
datanode	Runs a DFS datanode.
dfsadmin	Runs a DFS admin client.
mradmin	Runs a Map-Reduce admin client.
fsck	Runs a DFS filesystem checking utility.
fs	Runs a generic filesystem user client.
balancer	Runs a cluster balancing utility.
oiv	Applies the offline fsimage viewer to an fsimage.
fetchdt	Fetches a delegation token from the NameNode.
jobtracker	Runs the MapReduce job Tracker node.
pipes	Runs a Pipes job.
tasktracker	Runs a MapReduce task Tracker node.
historyserver	Runs job history servers as a standalone daemon.
job	Manipulates the MapReduce jobs.
queue	Gets information regarding JobQueues.
version	Prints the version.
jar <jar>	Runs a jar file.
distcp <srcurl> <desturl>	Copies file or directories recursively.
distcp2 <srcurl> <desturl>	DistCp version 2.
archive -archiveName NAME -p <parent path> <src>* <dest>	Creates a hadoop archive.
classpath	Prints the class path needed to get the Hadoop jar and the required libraries.
daemonlog	Get/Set the log level for each daemon

How to Interact with MapReduce Jobs

Usage: hadoop job [GENERIC_OPTIONS]

The following are the Generic Options available in a Hadoop job.

GENERIC_OPTIONS	Description
-submit <job-file>	Submits the job.

status <job-id>	Prints the map and reduce completion percentage and all job counters.
counter <job-id> <group-name> <countername>	Prints the counter value.
-kill <job-id>	Kills the job.
-events <job-id> <fromevent-#> <#-of-events>	Prints the events' details received by jobtracker for the given range.
-history [all] <jobOutputDir> - history < jobOutputDir>	Prints job details, failed and killed tip details. More details about the job such as successful tasks and task attempts made for each task can be viewed by specifying the [all] option.
-list[all]	Displays all jobs. -list displays only jobs which are yet to complete.
-kill-task <task-id>	Kills the task. Killed tasks are NOT counted against failed attempts.
-fail-task <task-id>	Fails the task. Failed tasks are counted against failed attempts.
set-priority <job-id> <priority>	Changes the priority of the job. Allowed priority values are VERY_HIGH, HIGH, NORMAL, LOW, VERY_LOW

To see the status of job

```
$ $HADOOP_HOME/bin/hadoop job -status <JOB-ID>
e.g.
$ $HADOOP_HOME/bin/hadoop job -status job_201310191043_0004
```

To see the history of job output-dir

```
$ $HADOOP_HOME/bin/hadoop job -history <DIR-NAME>
e.g.
$ $HADOOP_HOME/bin/hadoop job -history /user/expert/output
```

To kill the job

```
$ $HADOOP_HOME/bin/hadoop job -kill <JOB-ID>
e.g.
$ $HADOOP_HOME/bin/hadoop job -kill job_201310191043_0004
```

HADOOP - STREAMING

Hadoop streaming is a utility that comes with the Hadoop distribution. This utility allows you to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer.

Example Using Python

For Hadoop streaming, we are considering the word-count problem. Any job in Hadoop must have two phases: mapper and reducer. We have written codes for the mapper and the reducer in python script to run it under Hadoop. One can also write the same in Perl and Ruby.

Mapper Phase Code

```
!/usr/bin/python
import sys
# Input takes from standard input for myline in sys.stdin:
```

```
# Remove whitespace either side myline = myline.strip()
# Break the line into words words = myline.split()
# Iterate the words list for myword in words:
# Write the results to standard output print '%s\t%s' % (myword, 1)
```

Make sure this file has execution permission $chmod + x /home/expert/hadoop - 1.2.1/mapper.py$.

Reducer Phase Code

```
#!/usr/bin/python
from operator import itemgetter
import sys
current_word = ""
current_count = 0
word = ""
# Input takes from standard input for myline in sys.stdin:
# Remove whitespace either side myline = myline.strip()
# Split the input we got from mapper.py word, count = myline.split('\t', 1)
# Convert count variable to integer
   try:
      count = int(count)
except ValueError:
   # Count was not a number, so silently ignore this line continue
if current_word == word:
   current_count += count
else:
   if current_word:
      # Write result to standard output print '%s\t%s' % (current_word, current_count)
   current_count = count
   current_word = word
# Do not forget to output the last word if needed!
if current_word == word:
   print '%s\t%s' % (current_word, current_count)
```

Save the mapper and reducer codes in mapper.py and reducer.py in Hadoop home directory. Make sure these files have execution permission $chmod + xmapper.pyandchmod + xreducer.py$. As python is indentation sensitive so the same code can be download from the below link.

Execution of WordCount Program

```
$ $HADOOP_HOME/bin/hadoop jar contrib/streaming/hadoop-streaming-1.
2.1.jar \
   -input input_dirs \
   -output output_dir \
   -mapper <path/mapper.py \
   -reducer <path/reducer.py
```

Where "\" is used for line continuation for clear readability.

For Example,

```
./bin/hadoop jar contrib/streaming/hadoop-streaming-1.2.1.jar -input myinput -output
myoutput -mapper /home/expert/hadoop-1.2.1/mapper.py -reducer
/home/expert/hadoop-1.2.1/reducer.py
```

How Streaming Works

In the above example, both the mapper and the reducer are python scripts that read the input from standard input and emit the output to standard output. The utility will create a Map/Reduce job, submit the job to an appropriate cluster, and monitor the progress of the job until it completes.

When a script is specified for mappers, each mapper task will launch the script as a separate process when the mapper is initialized. As the mapper task runs, it converts its inputs into lines and feed the lines to the standard input *STDIN* of the process. In the meantime, the mapper collects the line-oriented outputs from the standard output *STDOUT* of the process and converts each line into a

key/value pair, which is collected as the output of the mapper. By default, the prefix of a line up to the first tab character is the key and the rest of the line *excludingthetabcharacter* will be the value. If there is no tab character in the line, then the entire line is considered as the key and the value is null. However, this can be customized, as per one need.

When a script is specified for reducers, each reducer task will launch the script as a separate process, then the reducer is initialized. As the reducer task runs, it converts its input key/values pairs into lines and feeds the lines to the standard input *STDIN* of the process. In the meantime, the reducer collects the line-oriented outputs from the standard output *STDOUT* of the process, converts each line into a key/value pair, which is collected as the output of the reducer. By default, the prefix of a line up to the first tab character is the key and the rest of the line *excludingthetabcharacter* is the value. However, this can be customized as per specific requirements.

Important Commands

Parameters	Options	Description
-input directory/file-name	Required	Input location for mapper.
-output directory-name	Required	Output location for reducer.
-mapper executable or script or JavaClassName	Required	Mapper executable.
-reducer executable or script or JavaClassName	Required	Reducer executable.
-file file-name	Optional	Makes the mapper, reducer, or combiner executable available locally on the compute nodes.
-inputformat JavaClassName	Optional	Class you supply should return key/value pairs of Text class. If not specified, TextInputFormat is used as the default.
-outputformat JavaClassName	Optional	Class you supply should take key/value pairs of Text class. If not specified, TextOutputformat is used as the default.
-partitioner JavaClassName	Optional	Class that determines which reduce a key is sent to.
-combiner streamingCommand or JavaClassName	Optional	Combiner executable for map output.
-cmdenv name=value	Optional	Passes the environment variable to streaming commands.
-inputreader	Optional	For backwards-compatibility: specifies a record reader class *insteadofaninputformatclass*.
-verbose	Optional	Verbose output.
-lazyOutput	Optional	Creates output lazily. For example, if the output format is based on FileOutputFormat, the output file is created only on the first call to output.collect *orContext. write*.
-numReduceTasks	Optional	Specifies the number of reducers.
-mapdebug	Optional	Script to call when map task fails.
-reducedebug	Optional	Script to call when reduce task fails.

HADOOP - MULTI NODE CLUSTER

This chapter explains the setup of the Hadoop Multi-Node cluster on a distributed environment.

As the whole cluster cannot be demonstrated, we are explaining the Hadoop cluster environment using three systems *onemasterandtwoslaves*; given below are their IP addresses.

- Hadoop Master: 192.168.1.15 $hadoop - master$
- Hadoop Slave: 192.168.1.16 $hadoop - slave - 1$
- Hadoop Slave: 192.168.1.17 $hadoop - slave - 2$

Follow the steps given below to have Hadoop Multi-Node cluster setup.

Installing Java

Java is the main prerequisite for Hadoop. First of all, you should verify the existence of java in your system using “java -version”. The syntax of java version command is given below.

```
$ java -version
```

If everything works fine it will give you the following output.

```
java version "1.7.0_71"
Java(TM) SE Runtime Environment (build 1.7.0_71-b13)
Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode)
```

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

Step 1

Download java (JDK - X64.tar.gz) by visiting the following link
http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

Then **jdk-7u71-linux-x64.tar.gz** will be downloaded into your system.

Step 2

Generally you will find the downloaded java file in Downloads folder. Verify it and extract the **jdk-7u71-linux-x64.gz** file using the following commands.

```
$ cd Downloads/
$ ls
jdk-7u71-Linux-x64.gz
$ tar zxf jdk-7u71-Linux-x64.gz
$ ls
jdk1.7.0_71 jdk-7u71-Linux-x64.gz
```

Step 3

To make java available to all the users, you have to move it to the location “/usr/local/”. Open the root, and type the following commands.

```
$ su
password:
# mv jdk1.7.0_71 /usr/local/
# exit
```

Step 4

For setting up **PATH** and **JAVA_HOME** variables, add the following commands to **~/.bashrc** file.

```
export JAVA_HOME=/usr/local/jdk1.7.0_71
export PATH=PATH:$JAVA_HOME/bin
```

Now verify the **java -version** command from the terminal as explained above. Follow the above process and install java in all your cluster nodes.

Creating User Account

Create a system user account on both master and slave systems to use the Hadoop installation.

```
# useradd hadoop
# passwd hadoop
```

Mapping the nodes

You have to edit **hosts** file in **/etc/** folder on all nodes, specify the IP address of each system followed by their host names.

```
# vi /etc/hosts
enter the following lines in the /etc/hosts file.
192.168.1.109 hadoop-master
192.168.1.145 hadoop-slave-1
192.168.56.1 hadoop-slave-2
```

Configuring Key Based Login

Setup ssh in every node such that they can communicate with one another without any prompt for password.

```
# su hadoop
$ ssh-keygen -t rsa
$ ssh-copy-id -i ~/.ssh/id_rsa.pub tutorialspoint@hadoop-master
$ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop_tp1@hadoop-slave-1
$ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop_tp2@hadoop-slave-2
$ chmod 0600 ~/.ssh/authorized_keys
$ exit
```

Installing Hadoop

In the Master server, download and install Hadoop using the following commands.

```
# mkdir /opt/hadoop
# cd /opt/hadoop/
# wget http://apache.mesi.com.ar/hadoop/common/hadoop-1.2.1/hadoop-1.2.0.tar.gz
# tar -xzf hadoop-1.2.0.tar.gz
# mv hadoop-1.2.0 hadoop
# chown -R hadoop /opt/hadoop
# cd /opt/hadoop/hadoop/
```

Configuring Hadoop

You have to configure Hadoop server by making the following changes as given below.

core-site.xml

Open the **core-site.xml** file and edit it as shown below.

```
<configuration>
   <property>
      <name>fs.default.name</name>
      <value>hdfs://hadoop-master:9000/</value>
   </property>
   <property>
      <name>dfs.permissions</name>
```

```
      <value>false</value>
   </property>
</configuration>
```

hdfs-site.xml

Open the **hdfs-site.xml** file and edit it as shown below.

```
<configuration>
   <property>
      <name>dfs.data.dir</name>
      <value>/opt/hadoop/hadoop/dfs/name/data</value>
      <final>true</final>
   </property>

   <property>
      <name>dfs.name.dir</name>
      <value>/opt/hadoop/hadoop/dfs/name</value>
      <final>true</final>
   </property>

   <property>
      <name>dfs.replication</name>
      <value>1</value>
   </property>
</configuration>
```

mapred-site.xml

Open the **mapred-site.xml** file and edit it as shown below.

```
<configuration>
   <property>
      <name>mapred.job.tracker</name>
      <value>hadoop-master:9001</value>
   </property>
</configuration>
```

hadoop-env.sh

Open the **hadoop-env.sh** file and edit JAVA_HOME, HADOOP_CONF_DIR, and HADOOP_OPTS as shown below.

Note: Set the JAVA_HOME as per your system configuration.

```
export JAVA_HOME=/opt/jdk1.7.0_17 export HADOOP_OPTS=-Djava.net.preferIPv4Stack=true
export HADOOP_CONF_DIR=/opt/hadoop/hadoop/conf
```

Installing Hadoop on Slave Servers

Install Hadoop on all the slave servers by following the given commands.

```
# su hadoop
$ cd /opt/hadoop
$ scp -r hadoop hadoop-slave-1:/opt/hadoop
$ scp -r hadoop hadoop-slave-2:/opt/hadoop
```

Configuring Hadoop on Master Server

Open the master server and configure it by following the given commands.

```
# su hadoop
$ cd /opt/hadoop/hadoop
```

Configuring Master Node

```
$ vi etc/hadoop/masters
hadoop-master
```

Configuring Slave Node

```
$ vi etc/hadoop/slaves
hadoop-slave-1
hadoop-slave-2
```

Format Name Node on Hadoop Master

```
# su hadoop
$ cd /opt/hadoop/hadoop
$ bin/hadoop namenode –format
```

```
11/10/14 10:58:07 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = hadoop-master/192.168.1.109
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 1.2.0
STARTUP_MSG: build = https://svn.apache.org/repos/asf/hadoop/common/branches/branch-1.2 -
r 1479473; compiled by 'hortonfo' on Mon May 6 06:59:37 UTC 2013
STARTUP_MSG: java = 1.7.0_71
************************************************************/ 11/10/14
10:58:08 INFO util.GSet: Computing capacity for map BlocksMap
editlog=/opt/hadoop/hadoop/dfs/name/current/edits
……….
……….
……. 11/10/14 10:58:08 INFO common.Storage: Storage directory
/opt/hadoop/hadoop/dfs/name has been successfully formatted. 11/10/14 10:58:08 INFO
namenode.NameNode:
SHUTDOWN_MSG: /************************************************************
SHUTDOWN_MSG: Shutting down NameNode at hadoop-master/192.168.1.15
************************************************************/
```

Starting Hadoop Services

The following command is to start all the Hadoop services on the Hadoop-Master.

```
$ cd $HADOOP_HOME/sbin
$ start-all.sh
```

Adding a New DataNode in the Hadoop Cluster

Given below are the steps to be followed for adding new nodes to a Hadoop cluster.

Networking

Add new nodes to an existing Hadoop cluster with some appropriate network configuration.
Assume the following network configuration.

For New node Configuration:

```
IP address : 192.168.1.103
netmask : 255.255.255.0
hostname : slave3.in
```

Adding User and SSH Access

Add a User

On a new node, add "hadoop" user and set password of Hadoop user to "hadoop123" or anything you want by using the following commands.

```
useradd hadoop
passwd hadoop
```

Setup Password less connectivity from master to new slave.

Execute the following on the master

```
mkdir -p $HOME/.ssh
chmod 700 $HOME/.ssh
ssh-keygen -t rsa -P '' -f $HOME/.ssh/id_rsa
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
chmod 644 $HOME/.ssh/authorized_keys
Copy the public key to new slave node in hadoop user $HOME directory
scp $HOME/.ssh/id_rsa.pub hadoop@192.168.1.103:/home/hadoop/
```

Execute the following on the slaves

Login to hadoop. If not, login to hadoop user.

```
su hadoop ssh -X hadoop@192.168.1.103
```

Copy the content of public key into file **"$HOME/.ssh/authorized_keys"** and then change the permission for the same by executing the following commands.

```
cd $HOME
mkdir -p $HOME/.ssh
chmod 700 $HOME/.ssh
cat id_rsa.pub >>$HOME/.ssh/authorized_keys
chmod 644 $HOME/.ssh/authorized_keys
```

Check ssh login from the master machine. Now check if you can ssh to the new node without a password from the master.

```
ssh hadoop@192.168.1.103 or hadoop@slave3
```

Set Hostname of New Node

You can set hostname in file **/etc/sysconfig/network**

```
On new slave3 machine
NETWORKING=yes
HOSTNAME=slave3.in
```

To make the changes effective, either restart the machine or run hostname command to a new machine with the respective hostname *restartisagoodoption*.

On slave3 node machine:

hostname slave3.in

Update **/etc/hosts** on all machines of the cluster with the following lines:

```
192.168.1.102 slave3.in slave3
```

Now try to ping the machine with hostnames to check whether it is resolving to IP or not.

On new node machine:

```
ping master.in
```

Start the DataNode on New Node

Start the datanode daemon manually using **$HADOOP_HOME/bin/hadoop-daemon.sh script**. It will automatically contact the master *NameNode* and join the cluster. We should also add the new node to the conf/slaves file in the master server. The script-based commands will recognize the new node.

Login to new node

```
su hadoop or ssh -X hadoop@192.168.1.103
```

Start HDFS on a newly added slave node by using the following command

```
./bin/hadoop-daemon.sh start datanode
```

Check the output of jps command on a new node. It looks as follows.

```
$ jps
7141 DataNode
10312 Jps
```

Removing a DataNode from the Hadoop Cluster

We can remove a node from a cluster on the fly, while it is running, without any data loss. HDFS provides a decommissioning feature, which ensures that removing a node is performed safely. To use it, follow the steps as given below:

Step 1

Login to master.

Login to master machine user where Hadoop is installed.

```
$ su hadoop
```

Step 2

Change cluster configuration.

An exclude file must be configured before starting the cluster. Add a key named dfs.hosts.exclude to our **$HADOOP_HOME/etc/hadoop/hdfs-site.xml** file. The value associated with this key provides the full path to a file on the NameNode's local file system which contains a list of machines which are not permitted to connect to HDFS.

For example, add these lines to **etc/hadoop/hdfs-site.xml** file.

```
<property>
   <name>dfs.hosts.exclude</name>
   <value>/home/hadoop/hadoop-1.2.1/hdfs_exclude.txt</value>
   <description>DFS exclude</description>
</property>
```

Step 3

Determine hosts to decommission.

Each machine to be decommissioned should be added to the file identified by the hdfs_exclude.txt,

one domain name per line. This will prevent them from connecting to the NameNode. Content of the **"/home/hadoop/hadoop-1.2.1/hdfs_exclude.txt"** file is shown below, if you want to remove DataNode2.

```
slave2.in
```

Step 4

Force configuration reload.

Run the command **"$HADOOP_HOME/bin/hadoop dfsadmin -refreshNodes"** without the quotes.

```
$ $HADOOP_HOME/bin/hadoop dfsadmin -refreshNodes
```

This will force the NameNode to re-read its configuration, including the newly updated ‘excludes’ file. It will decommission the nodes over a period of time, allowing time for each node's blocks to be replicated onto machines which are scheduled to remain active.

On **slave2.in**, check the jps command output. After some time, you will see the DataNode process is shutdown automatically.

Step 5

Shutdown nodes.

After the decommission process has been completed, the decommissioned hardware can be safely shut down for maintenance. Run the report command to dfsadmin to check the status of decommission. The following command will describe the status of the decommission node and the connected nodes to the cluster.

```
$ $HADOOP_HOME/bin/hadoop dfsadmin -report
```

Step 6

Edit excludes file again.

Once the machines have been decommissioned, they can be removed from the ‘excludes’ file. Running **"$HADOOP_HOME/bin/hadoop dfsadmin -refreshNodes"** again will read the excludes file back into the NameNode; allowing the DataNodes to rejoin the cluster after the maintenance has been completed, or additional capacity is needed in the cluster again, etc.

Special Note: If the above process is followed and the tasktracker process is still running on the node, it needs to be shut down. One way is to disconnect the machine as we did in the above steps. The Master will recognize the process automatically and will declare as dead. There is no need to follow the same process for removing the tasktracker because it is NOT much crucial as compared to the DataNode. DataNode contains the data that you want to remove safely without any loss of data.

The tasktracker can be run/shutdown on the fly by the following command at any point of time.

```
$ $HADOOP_HOME/bin/hadoop-daemon.sh stop tasktracker $HADOOP_HOME/bin/hadoop-daemon.sh start tasktracker
```