

## UNIT – 2

### WORKING WITH BIG DATA

#### **INTRODUCTION TO BIG DATA:**

Big Data is data that exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or doesn't fit the strictures [restrictions] of our existing database architectures. To gain value from this data, you must choose an alternative way to process it.

The hot IT buzzword of 2012, Big Data has become viable as cost-effective approaches have emerged to tame the Volume, Velocity and Variability of massive data. Within this data lie valuable patterns and information, previously hidden because of the amount of work required to extract them. To leading corporations, like Walmart or Google, this power has been in reach for some time, but at fantastic cost. Today's commodity hardware, cloud architectures and open source software bring big data processing into the reach of the less well-resourced. Big data processing is eminently feasible for even the small garage startups, who can cheaply rent server time in the cloud.

The value of big data to an organization falls into two categories: i) analytical use and ii) enabling new products. Big data analytics can reveal insights hidden previously by data too costly to process, such as peer influence among customers, revealed by analyzing shopper's transactions, social and geographical data. Being able to process every item of data in reasonable time removes the troublesome need for sampling and promotes an investigative approach to data, in contrast to the somewhat static nature of running predetermined reports. The past decades successful web startups are prime examples of big data used as an enabler of new products and services. For example, by combining a large number of signals from a user's actions and those of their friends, Facebook has been able to craft a highly personalized user experience and create a new kind of advertising business. It's no coincidence that the lion's share of ideas and tools underpinning big data have emerged from Google, Yahoo, Amazon and Facebook.

#### **2.1. Google File System (GFS):**

Google File System (GFS or Google FS) is a proprietary distribute file system developed by Google for its own usage purpose. It is designed to provide efficient, reliable access to data using large clusters of commodity hardware.

GFS is enhanced for Google's core data storage and usage needs (primarily the search engine), which can generate enormous amounts of data that needs to be retained. Google File System grew out of an earlier Google effort, "BigFiles", developed by Larry Page and Sergey Brin in the early days of Google, while it was still located in Stanford. Files are divided into fixed size chunks of 64 megabytes, similar to clusters or sectors in regular file systems, which are only extremely rarely overwritten, or shrunk (become smaller); files are usually appended to or read. It is also designed and optimized to run on Google's computing clusters, dense nodes which consist of cheap "commodity" computers, which means precautions must be taken against the high failure rate of individual nodes and the subsequent data loss. Other design decisions select for high data throughputs, even when it comes at the cost of latency.

A GFS cluster consists of multiple nodes. These nodes are divided into 2 types: i) Master Node and ii) A large number of Chunkservers.

Each file is divided into fixed size chunks. Chunk servers store these chunks. Each chunk is assigned a unique 64bit label by the master node at the time of creation, and logical mappings of files to constituent chunks are maintained. Each chunk is replicated several times throughout the network, with the minimum being three, but even more for files that have high end-in demand or need more redundancy. The Master server does not usually store the actual chunks, but rather all the metadata associated with the chunks, such as the tables mapping the 64bit labels to chunk locations and the files they make up, the locations of the copies of the chunks, what processes are reading or writing to a particular chunk, or taking a "snapshot" of the chunk pursuant to replicate it (usually at the

instigation of the Master server, when, due to node failures, the number of copies of a chunk has fallen beneath the set number). All this metadata is kept current by the Master server periodically receiving updates from each chunk server ("Heartbeat messages"). Permissions for modifications are handled by a system of time-limited, expiring "leases", where the Master server grants permission to a process for a finite period of time during which no other process will be granted permission by the Master server to modify the chunk. The modifying chunkserver, which is always the primary chunk holder, then propagates the changes to the chunkservers with the backup copies. The changes are not saved until all chunkservers acknowledge, thus guaranteeing the completion and atomicity of the operation. Programs access the chunks by first querying the Master server for the locations of the desired chunks; if the chunks are not being operated on (i.e. no outstanding leases exist), the Master replies with the locations, and the program then contacts and receives the data from the chunkserver directly (similar to Kazaa and its supernodes). Unlike most other file systems, GFS is not implemented in the kernel of an operating system, but is instead provided as a userspace library.

The below Fig. 2.1 Shows the GFS architecture.

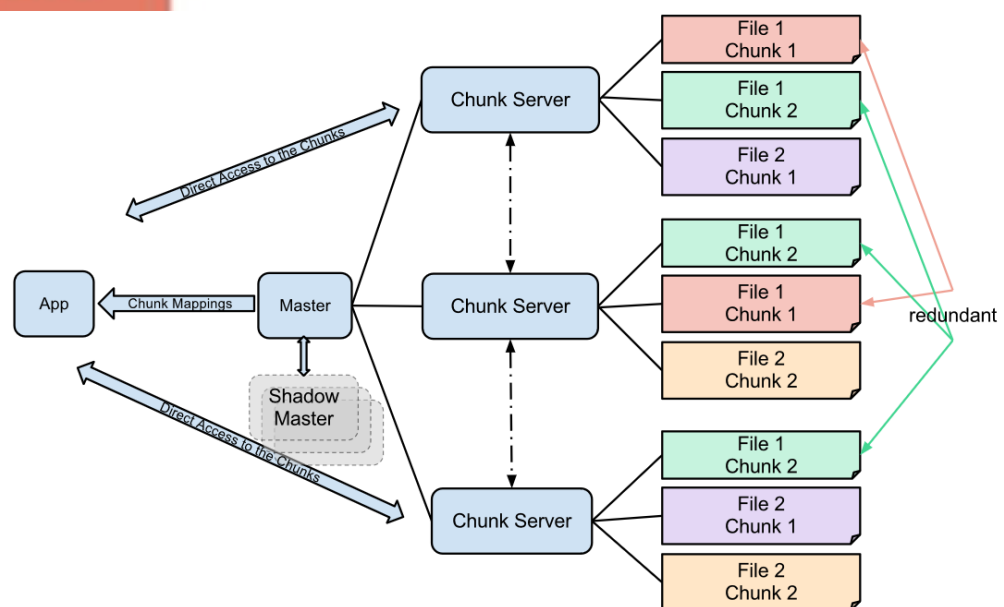
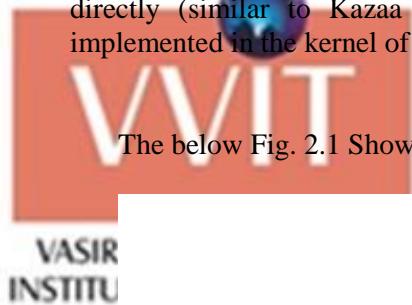


Fig.2.1: Google File System Architecture

## **HADOOP - INTRODUCTION & OVERVIEW:**

Big Data is a collection of datasets so large and complex that it becomes difficult to store and process using on-hand database management tools or traditional data processing applications. Big Data Analytics consists the following:

- Examining large amount of data.
- Appropriate information.
- Identification of hidden patterns, unknown correlations.
- Competitive advantage.
- Better business decisions: strategic and operational.
- Effective marketing, customer satisfaction, increased revenue.

To do analysis with big data, we need some tools which are available in real time. Out of them, Hadoop was became as popular, important and effective tools for big data analytics.

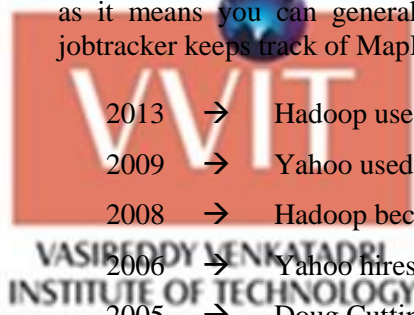
## **BRIEF HISTORY OF HADOOP:**

Hadoop was created by Doug Cutting, the creator of Apache Lucene, the widely used text search library. Hadoop has its origins in Apache Nutch, an open source web search engine, itself a part of the Lucene project.

## The Origin of the Name “Hadoop”:

The name Hadoop is not an acronym; it's a made-up name. The project's creator, Doug Cutting, explains how the name came about:

The name his kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are my naming criteria. Kids are good at generating such. Googol is a kid's term. Subprojects and “contrib” modules in Hadoop also tend to have names that are unrelated to their function, often with an elephant or other animal theme (“Pig,” for example). Smaller components are given more descriptive (and therefore more mundane) names. This is a good principle, as it means you can generally work out what something does from its name. For example, the jobtracker keeps track of MapReduce jobs.

- 
- 2013 → Hadoop used by Hundreds of Companies.
  - 2009 → Yahoo used Hadoop to sort 1 TB in 62 Secs.
  - 2008 → Hadoop become Apache Top Level Project.
  - 2006 → Yahoo hires Doug Cutting to work on Hadoop with a dedicated team.
  - 2005 → Doug Cutting & Nutch team implemented Google's frameworks in Nutch.
  - 2004 → Google publishes Google File System (GFS) & Map Reduce Framework Papers.

The core Hadoop contains Two components: i) HDFS (Hadoop Distributed File System)  
ii) MapReduce (Distributed Data Processing Model)

## Advantages of Hadoop:

Some of the advantages of Hadoop was listed below:

- Accessible
- Available
- Scalable
- Reliable
- Robust
- Simple
- Cost Effective
- Flexible
- Fast and Resilient to Failures

## Apache Hadoop Ecosystem:

The Hadoop Ecosystem consists of several Hadoop Projects which were given below:

### *i) Common:*

A set of components and interfaces for distributed filesystems and general I/O (serialization, Java RPC, persistent data structures).

### *ii) Avro:*

A serialization system for efficient, cross-language RPC, and persistent data storage.

### *iii) MapReduce:*

A distributed data processing model and execution environment that runs on large clusters of commodity machines.

**iv) HDFS:**

A distributed filesystem that runs on large clusters of commodity machines.

**v) Pig:**

A data flow language and execution environment for exploring very large datasets. Pig runs on HDFS and MapReduce clusters.

**vi) Hive:**

A distributed data warehouse. Hive manages data stored in HDFS and provides a query language based on SQL (and which is translated by the runtime engine to MapReduce jobs) for querying the data.

**vii) HBase:**

A distributed, column-oriented database. HBase uses HDFS for its underlying storage, and supports both batch-style computations using MapReduce and point queries (random reads).

**viii) ZooKeeper:**

A distributed, highly available coordination service. ZooKeeper provides primitives such as distributed locks that can be used for building distributed applications.

**ix) Sqoop:**

A tool for efficiently moving data between relational databases and HDFS.

**x) Oozie:**

A Service for running and scheduling workflows of Hadoop job (including MapReduce, Pig, Hive and Sqoop jobs).

**xi) Flume:**

A distributed, reliable and available service for efficiently collecting, aggregating and moving large amounts of log data.

**Integrations:**

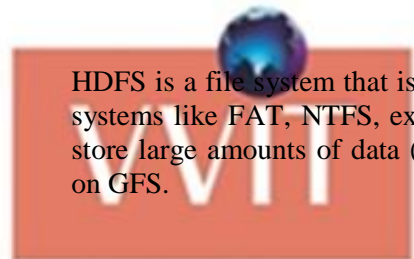
- Hive  $\leftrightarrow$  Hbase
- MapReduce  $\leftrightarrow$  Hive
- MapReduce  $\leftrightarrow$  Hbase

## **2.2. Hadoop Distributed File System (HDFS):**

Hadoop is made up of 2 parts:

1. HDFS – Hadoop Distributed File System
2. MapReduce – The programming model that is used to work on the data present in HDFS.

### **HDFS – Hadoop Distributed File System**



HDFS is a file system that is written in Java and resides within the user space unlike traditional file systems like FAT, NTFS, ext2, etc that reside on the kernel space. HDFS was primarily written to store large amounts of data (terabytes and petabytes). HDFS was built inline with Google's paper on GFS.

### **MapReduce** DY VENKATADRI INSTITUTE OF TECHNOLOGY

MapReduce is the programming model that uses Java as the programming language to retrieve data from files stored in the HDFS. All data in HDFS is stored as files. Even MapReduce was built inline with another paper by Google.

Google, apart from their papers did not release their implementations of GFS and MapReduce. However, the Open Source Community built Hadoop and MapReduce based on those papers. The initial adoption of Hadoop was at Yahoo Inc., where it gained good momentum and went onto be a part of their production systems. After Yahoo, many organizations like LinkedIn, Facebook, Netflix and many more have successfully implemented Hadoop within their organizations.

Hadoop uses HDFS to store files efficiently in the cluster. When a file is placed in HDFS it is broken down into blocks, 64 MB block size by default. These blocks are then replicated across the different nodes (*DataNodes*) in the cluster. The default replication value is 3, i.e. there will be 3 copies of the same block in the cluster. We will see later on why we maintain replicas of the blocks in the cluster.

A Hadoop cluster can comprise of a single node (single node cluster) or thousands of nodes.

Once you have installed Hadoop you can try out the following few basic commands to work with

### **Basic HDFS Commands:**

```
hadoop fs -ls
```

```
hadoop fs -put <path_of_local> <path_in_hdfs>
```

```
hadoop fs -get <path_in_hdfs> <path_of_local>
```

```
hadoop fs -cat <path_of_file_in_hdfs>
```

```
hadoop fs -rmr <path_in_hdfs>
```

The different components of a Hadoop Cluster are:

NameNode (Master) – NameNode, Secondary NameNode, JobTracker

DataNode 1 (Slave) – TaskTracker,

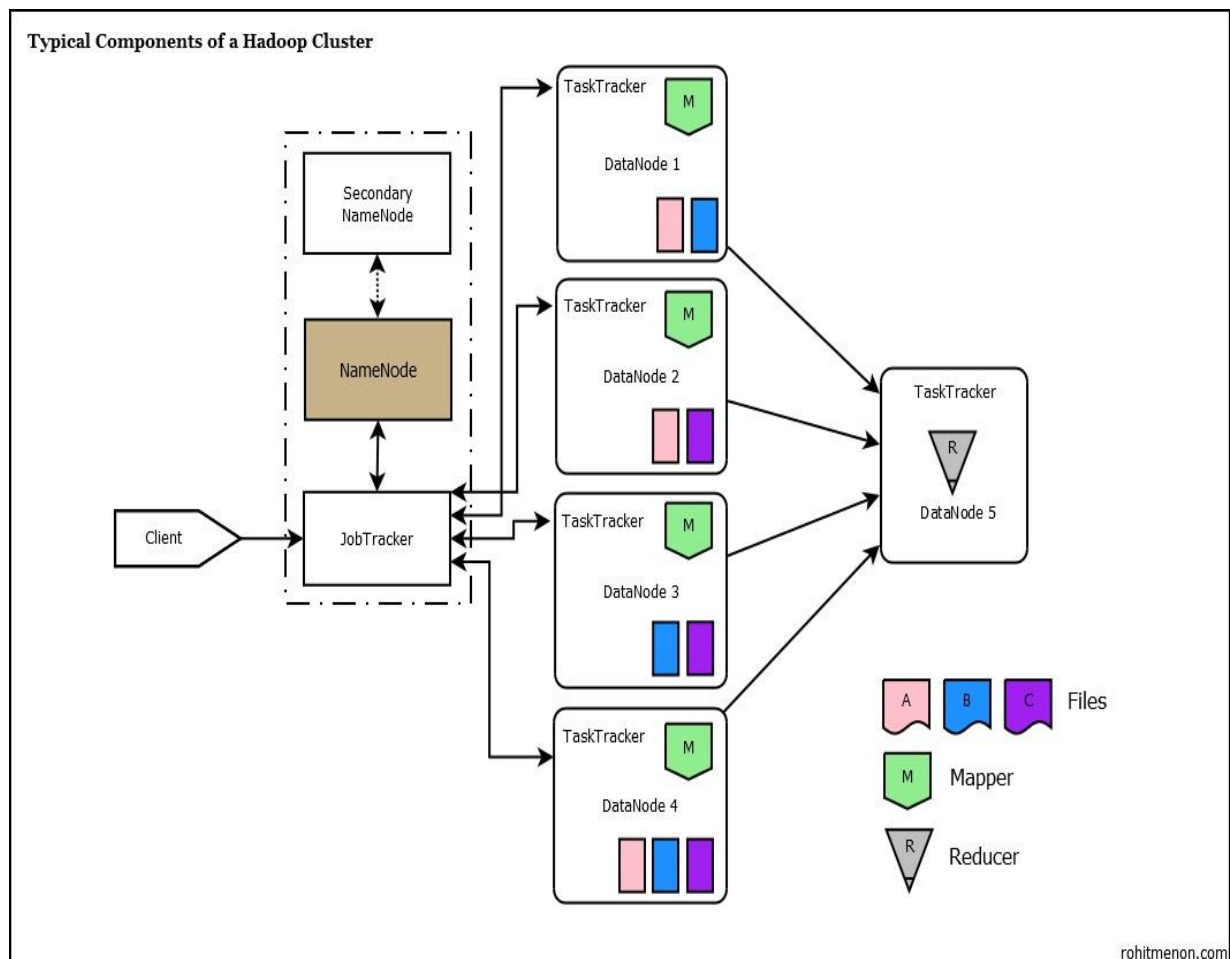
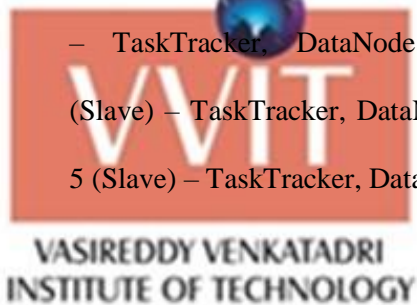
DataNode 2 (Slave) –

TaskTracker, DataNode 3 (Slave)

– TaskTracker, DataNode 4

(Slave) – TaskTracker, DataNode

5 (Slave) – TaskTracker, DataNode



The above diagram depicts a 6 Node Hadoop Cluster.



In the diagram we will see that the *NameNode*, *Secondary NameNode* and the *JobTracker* are running on a single machine. Usually in production clusters having more than 20-30 nodes, the daemons run on separate nodes.

Hadoop follows a Master-Slave architecture. As mentioned earlier, a file in HDFS is split into blocks and replicated across *Datanodes* in a Hadoop cluster. You can see that the three files A, B and C have been split across with a replication factor of 3 across the different *Datanodes*.

Now let us go through each node and daemon:

### **NameNode**

The *NameNode* in Hadoop is the node where Hadoop stores all the location information of the files in HDFS. In other words, it holds the metadata for HDFS. Whenever a file is placed in the cluster a corresponding entry of its location is maintained by the *NameNode*. So, for the files A, B and C we would have something as follows in the *NameNode*:

File A – *DataNode1*, *DataNode2*, *DataNode4*

File B – *DataNode1*, *DataNode3*, *DataNode4*

File C – *DataNode2*, *DataNode3*, *DataNode4*

This information is required when retrieving data from the cluster as the data is spread across multiple machines. The *NameNode* is a Single Point of Failure for the Hadoop Cluster.

### **Secondary NameNode**

IMPORTANT – The *Secondary NameNode* is not a failover node for the *NameNode*.

The secondary name node is responsible for performing periodic housekeeping functions for the *NameNode*. It only creates checkpoints of the file system present in the *NameNode*.

### **DataNode**

The *DataNode* is responsible for storing the files in HDFS. It manages the file blocks within the node. It sends information to the *NameNode* about the files and blocks stored in that node and responds to the *NameNode* for all filesystem operations.

### **JobTracker**

*JobTracker* is responsible for taking in requests from a client and assigning *TaskTrackers* with tasks to be performed. The *JobTracker* tries to assign tasks to the *TaskTracker* on the *DataNode* where the data is locally present (Data Locality). If that is not possible it will at least try to assign tasks to *TaskTrackers* within the same rack. If for some reason the node fails the *JobTracker* assigns the task to another *TaskTracker* where the replica of the data exists since the data blocks are replicated across the *Datanodes*. This ensures that the job does not fail even if a node fails within the cluster.

### **TaskTracker**

*TaskTracker* is a daemon that accepts tasks (Map, Reduce and Shuffle) from the *JobTracker*. The *TaskTracker* keeps sending a heart beat message to the *JobTracker* to notify that it is alive. Along with the heartbeat it also sends the free slots available within it to process

tasks. *TaskTracker* starts and monitors the Map & Reduce Tasks and sends progress/status information back to the *JobTracker*.

All the above daemons run within have their own JVMs. A typical (simplified) flow in Hadoop is as follows:

1. A Client (usually a MapReduce program) submits a job to the *JobTracker*.
2. The *JobTracker* gets information from the *NameNode* on the location of the data within the *DataNodes*. The *JobTracker* places the client program (usually a jar file along with the configuration file) in the HDFS. Once placed, *JobTracker* tries to assign tasks to *TaskTrackers* on the *DataNodes* based on data locality.
3. The *TaskTracker* takes care of starting the Map tasks on the *DataNodes* by picking up the client program from the shared location on the HDFS.
4. The progress of the operation is relayed back to the *JobTracker* by the *TaskTracker*.
5. On completion of the Map task an intermediate file is created on the local filesystem of the *TaskTracker*.
6. Results from Map tasks are then passed on to the Reduce task.
7. The Reduce tasks work on all data received from map tasks and write the final output to HDFS.
8. After the task completes the intermediate data generated by the *TaskTracker* is deleted.

A very important feature of Hadoop to note here is, that, the program goes to where the data is and not the way around, thus resulting in efficient processing of data.

### **2.3. Introducing and Configuring Hadoop Cluster [Local, Pseudo-Distributed, Fully Distributed Mode] & Configuring XML Files:**

Apache Hadoop is an open source framework for storing and distributed batch processing of huge datasets on clusters of commodity hardware. Hadoop can be used on a single machine (Standalone Mode) as well as on a cluster of machines (Distributed Mode – Pseudo & Fully). One of the striking features of Hadoop is that it efficiently distributes large amounts of work across a cluster of machines/commodity hardware.

#### **2.3.1. Configuring Hadoop Cluster:**

Before configuring Hadoop Cluster, it is important to understand that Hadoop can be run in any of the following 3 modes:

##### **i) Standalone Mode:**

In standalone mode, we will configure Hadoop on a single machine (e.g. an Ubuntu machine on the host VM). The configuration in standalone mode is quite straightforward and does not require major changes.



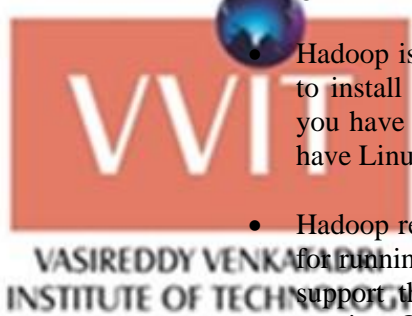
ii) **Pseudo-Distributed Mode:**

In a pseudo distributed environment, we will configure more than one machine, one of these to act as a master and the rest as slave machines/node. In addition we will have more than one Ubuntu machine playing on the host VM.

iii) **Fully Distributed Mode:**

Fully Distributed Mode is quite similar to a pseudo distributed environment with the exception that instead of VM the machines/node will be on a real distributed environment.

To configure Hadoop, we need some pre-requisites which are given below:



- 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.
- Hadoop requires Java 1.5+ installation. However, using Java 1.6 is recommended for running Hadoop. It can be run on both Windows & Unix but Linux/Unix best support the production environment. Working with Hadoop on Windows also requires Cygwin installation

### 2.3.1.1 Installing & Configuring Hadoop in Standalone Mode:

We might want to create a dedicated user for running Apache Hadoop, but it is not a mandatory prerequisite even it is recommended. Now, we will see the creation of user and also using of default user for running Hadoop.

#### **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
```

(Or) we will also use a default user for running Hadoop.

#### **Environment:**

Ubuntu 10.10  
JDK 6 or above  
Hadoop1.1.2 (Any stable release)

Follow these steps for installing and configuring Hadoop on a single node:

### **Step1. Install Java**

We will use Java 1.6, the installation as follows.

Use the below command to begin the installation of Java

```
1
$ sudo apt-get install openjdk-6-jdk
```

Or

This will install the full JDK under /usr/lib/jvm/java-6-sun directory.

### **Step2. Verify Java installation**

You can verify java installation using the following command

```
1
$ java -version
```

On executing this command, you should see output similar to the following:

```
java version "1.6.0_27"
Java(TM) SE Runtime Environment (build 1.6.0_45b06)
Java HotSpot(TM) 64Bit Server VM (build 20.45b01, mixed mode)
```

### **Step3. SSH configuration**

```
1
sudo apt-get install ssh
```

Generate ssh key

ssh-keygen -t rsa -P "" (press enter when asked for a file name; this will generate a passwordless ssh file)

Now copy the public key (id\_rsa.pub) of current machine to authorized\_keys. Below command copies the generated public key in the .ssh/authorized\_keys file:

```
1
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

```
1
ssh localhost
```

Pressing yes will add localhost to known hosts

### **Step4. Download Hadoop**

Download the latest stable release of Apache Hadoop from <http://hadoop.apache.org/releases.html>.

Unpack the release tar – zxvf hadoop1.0.3.tar.gz

Save the extracted folder to an appropriate location, HADOOP\_HOME will be pointing to this directory.

### **Step5. Verify Hadoop**

Check if the following directories exist under HADOOP\_HOME: bin, conf, lib, bin

Use the following command to create an environment variable that points to the Hadoop installation directory (HADOOP\_HOME)

```
1
export HADOOP_HOME=/home/user/hadoop
```

Now place the Hadoop binary directory on your commandline path by executing the command

```
1
export PATH=$PATH:$HADOOP_HOME/bin
```

Use this command to verify your Hadoop installation:

```
hadoop version
```

The o/p should be similar to below one

Hadoop 1.1.2

Subversion <https://svn.apache.org/repos/asf/hadoop/common/branches/branch0.20>  
-r911707

Compiled by MICSE on Fri Jun 17 15:01:02 IST 2016

#### **Step6. Configure JAVA\_HOME**

Hadoop requires Java installation path to work on, for this we will be setting JAVA\_HOME environment variable and this will point to our Java installation dir.

Java\_Home can be configured in ~/.bash\_profile or ~/.bashrc file. Alternatively you can also let hadoop know this by setting Java\_Home in hadoop conf/hadoop-env.sh file.

Use the below command to set JAVA\_HOME on Ubuntu

```
1
export JAVA_HOME=/usr/lib/jvm/java-6-sun
JAVA_HOME can be verified by command
```

```
1
echo $JAVA_HOME
```

#### **Step7. Create Data Directory for Hadoop**

An advantage of using Hadoop is that with just a limited number of directories you can set it up to work correctly. Let us create a directory with the name hdfs and three sub-directories: name, data and tmp.

Since a Hadoop user would require to read-write to these directories you would need to change the permissions of above directories to 755 or 777 for Hadoop user.

#### **Step8. Configure Hadoop XML files**

Next, we will configure Hadoop XML file. Hadoop configuration files are in the HADOOP\_HOME/conf dir.

### **conf/coresite.xml**

```
<!--?xml version= "1.0" -->>
<!--?xml -stylesheet type= "text/xsl" href= "configuration.xsl" ?-->
<!-- Putting site-specific property overrides the file. -->
fs.default.name
hdfs://localhost:9000
hadoop.temp.dir
/home/miccse/hdfs/temp<span style= "font-family: Georgia, 'Times New Roman',
'Bitstream Charter', Times, serif; font-size: 13px; line-height: 19px;" > </span>
```

### **conf/hdfs-site.xml**

```
<!-- Putting site specific property overrides in the file. -->
dfs.name.dir /home/miccse/hdfs/name dfs.data.dir /home/miccse/hdfs/data
dfs.replication 1
```

```
<strong style= "font-family: Georgia, 'Times New Roman', 'Bitstream Charter', Times,
serif; font-size: 13px; line-height: 19px;" >conf/mapred-site.xml</strong>
```

1

2

3

4

```
<!-- Putting site-specific property overrides this file. -->
```

```
mapred.job.tracker
```

```
localhost: 9001
```

### **conf/masters**

Not required in single node cluster.

### **conf/slaves**

Not required in single node cluster.

### **Step9. Format Hadoop Name Node**

Execute the below command from hadoop home directory

1

```
$ ~/hadoop/bin/hadoop namenode -format
```

### **Step10. Start Hadoop daemons**

### **Step11. Verify the daemons are running**

1

```
$ jps ( if jps is not in path, try /usr/java/latest/bin/jps)
output will look similar to this
```

```
9316 SecondaryNameNode
```

```
9203 DataNode
```

```
9521 TaskTracker
```

```
9403 JobTracker
```

```
9089 NameNode
```

Now we have all the daemons running:

Note: If your master server fails to start due to the dfs safe mode issue, execute this on the Hadoop command line:

```
1
hadoop dfsadmin -safemode leave
```

Also make sure to format the namenode again if you make changes to your configuration.

#### **Step12. Verify UIs by namenode & job tracker**

Open a browser window and type the following URLs:

**namenode UI:** [http://machine\\_host\\_name:50070](http://machine_host_name:50070)

**job tracker UI:** [http://machine\\_host\\_name:50030](http://machine_host_name:50030)

substitute 'machine host name' with the public IP of your node e.g:  
<http://localhost:50070>

**Now you have successfully installed and configured Hadoop on a single node.**

#### **Basic Hadoop Admin Commands (Source: Getting Started with Hadoop):**

The ~/hadoop/bin directory contains some scripts used to launch Hadoop DFS and Hadoop Map/Reduce daemons. These are:

- startall.sh – Starts all Hadoop daemons, the namenode, datanodes, the jobtracker and tasktrackers.
- stopall.sh – Stops all Hadoop daemons.
- startmapred.sh – Starts the Hadoop Map/Reduce daemons, the jobtracker and tasktrackers.
- stopmapred.sh – Stops the Hadoop Map/Reduce daemons.
- startdfs.sh – Starts the Hadoop DFS daemons, the namenode and datanodes.
- stopdfs.sh – Stops the Hadoop DFS daemons.

### **2.3.1.2 Installing & Configuring Hadoop in Pseudo-Distributed Mode:**

In previous topic, we have discussed prerequisites of setting up Hadoop correctly and explained in detail how to setup Apache Hadoop in Standalone Mode. Now, we will illustrate the steps required to setup Apache Hadoop in Pseudo Distributed Mode. This mode has also consists all the steps which includes the updating of master, slave nodes configuration and xml files configuration.

#### **Step1: Configuring master & slave nodes**

We will be using two machines one as a master and the other one as slave. I have used Ubuntu11.10. In this demonstration. I named one virtual machine as Ubuntu1 and other as Ubuntu 2 respectively.

Change the hostname of these machines using the command:

```
1
$ sudo gedit /etc/hostname
```

Give Ubuntu1 hostname as master and Ubuntu2 as slave. We can verify the hostname by executing the command: `$ hostname`

If the updated name fails to appear then restart the hostname service using the command:

```
1
$ sudo service hostname start
```

```
1
$ sudo gedit /etc/hosts
```

Add master and slave machine IP & name IPs as 192.168.118.149 for master 192.168.118.151 for slave (For Example).

### **Step2: Configuring SSH on all nodes (master & slaves)**

Install SSH on all nodes using the commands.

```
1
$ sudo apt-get install ssh
(this step is required to connect to other machines).
```

Generate ssh key `ssh-keygen-t rsa -P ""` (press enter when asked for file name; this will generate a passwordless ssh file).

Now copy the public key (id\_rsa.pub) of current machine to authorized\_keys. Executing the following command will copy the generated public key in the .ssh/authorized\_keys file.

```
1
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

```
1
ssh localhost
```

Pressing yes will add localhost to known hosts

Once SSH is successfully configured on all nodes, confirm passwordless SSH connectivity from master to slave nodes and vice versa using this command.

```
1
ssh master
```

### **Step3: Install Java on all nodes (master & slaves)**

First check if Java is already installed or not by running the command

```
1
$ java -version
```

```
1
$ sudo apt-get install openjdk-6-jdk
```

Install the utility python softwares by running these commands

```
1
2
3
```

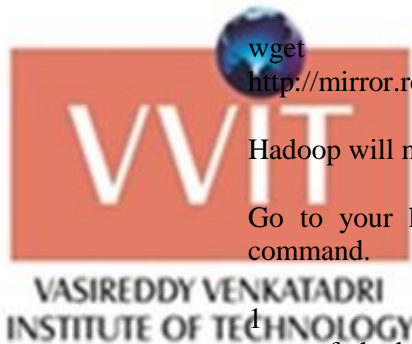


#### **Step4: Installing & Configuring Hadoop all nodes (master & slaves)**

Hadoop installation is required on all nodes, You may download a stable version of Hadoop from this link (Releases > Download> Download a release now)  
<http://start.ubuntu.com/11.04/Google/?sourceid=hp>

Or alternatively

Copy the link Location-hadoop-1.2.1.tar.gz (hadoop1.0.1.tar.gz)  
<http://mirror.reverse.net/pub/apache/hadoop/common/hadoop1.2.1/hadoop1.2.1.tar.gz>



wget  
<http://mirror.reverse.net/pub/apache/hadoop/common/hadoop1.2.1/hadoop1.2.1.tar.gz>

Hadoop will now get downloaded in the home folder

Go to your home folder and extract the downloaded Hadoop tar file using the command.

```
1 tar -xzf hadoop- 1.2 . 1 .tar.gz
HADOOP_HOME is in 1.2.1 version but if you are using any older versions you
might want to set HADOOP_HOME to the latest version using the commands.
```

```
1
2
3
4
sudo gedit ~/. bashrc (befor this you may require changing the file permission by: sudo
chmod 777 <filename> )
export HADOOP_HOME=/user/home/hadoop ( this is just an example)
export HADOOP_HOME = /home/miccse/hadoop- 1.2.1 (this is just an example)
export PATH= $PATH:$HADOOP_HOME/bin
```

Set Java\_Home in conf/hadoopenv.sh file as below.

Usually Java is installed at /usr/lib/, jvm Pick the sun java, right click on THIRDPARTYLICENSEREADME.txt and see the Location value, should look like /usr/lib/jvm/java-6-sun.

Change the Hadoop home directory privileges

```
1
chown -R miccse hadoop-1.2.1
(Instead of pasting it to terminal prefer typing)
```

```
1
chmod -R 755 hadoop- 1.2 . 1
```

#### **Step5: Modify Hadoop configuration files (master & slaves)**

- Create a dir hdfs with subdirectories as data, name and temp.
- Create a dir tempdir under home directory of the user (you want to use for hadoop)
- Update conf/coresite.xml as below:
- Change <name>hadoop.tmp.dir</name> value to /home/miccse/tempdir Like below:  
<value>/home/miccse/tempdir</value>

- Change localhost<name>fs.default.name</name><value>hdfs://localhost:54310</value> Like below:  
<value>hdfs://master:9000</value>

