##### Group(A)

**ASSIGNMENTNO.1**

##### ProblemStatement:

**Aim:  To Perform Hadoop Installation (Configuration) on  a)Single Node b)Multiple Node**

##### Tools/Environment:

Ubuntu

##### RelatedTheory:

Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common and should be automatically handled by the framework. The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part called MapReduce. Hadoop splits files into large blocks and distributes them across nodes in a cluster. To process data, Hadoop transfers packaged code for nodes to process in parallel based on the data that needs to be processed. This approach takes advantage of data locality— nodes manipulating the data they have access to— to allow the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking. The base Apache Hadoop framework is composed of the following modules:

* Hadoop Common – contains libraries and utilities needed by other Hadoop modules;
* Hadoop Distributed File System (HDFS) – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster;
* Hadoop YARN – a resource-management platform responsible for managing computing resources in clusters and using them for scheduling of users' applications; and
* HadoopMapReduce – an implementation of the MapReduce programming model for large scale data processing.

**a)Single Node:**

**Steps for Compilation & Execution**

sudo apt-get update

sudo apt-get install openjdk-7-jre-headless sudo apt-get install openjdk-7-jdk sudo apt-get install sshsudo apt-get install rsync

# Download hadoop from : http://www.eu.apache.org/dist/hadoop/common/stable/hadoop-

2.9.0.tar.gz

# copy and extract hadoop-2.9.0.tar.gz in home folder

# rename the name of the extracted folder from hadoop-2.9.0 to hadoopreadlink -f /usr/bin/javac

# find whether ubuntu is 32 bit (i686) or 64 bit (x86\_64) uname -i

gedit ~/hadoop/etc/hadoop/hadoop-env.sh

# add following line in it # for 32 bit ubuntu

export JAVA\_HOME=/usr/lib/jvm/java-7-openjdk-**i386**

# for 64 bit ubuntu

export JAVA\_HOME=/usr/lib/jvm/java-7-openjdk-**amd64**

# save and exit the file

# to display the usage documentation for the hadoop script try next command ~/hadoop/bin/hadoop

## 1. For standalone mode

mkdir input

cp ~/hadoop/etc/hadoop/\*.xml input

~/hadoop/bin/hadoop jar ~/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.9.0.jar grep input output 'us[a-z.]+' cat output/\*

# Our task is done, so remove input and output folders rm -r input output

## # 2. Pseudo-Distributed mode

# get your user name

whoami

# remember your user name, we'll use it in the next step gedit ~/hadoop/etc/hadoop/core-site.xml

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://localhost:1234</value>

</property>

</configuration>

gedit ~/hadoop/etc/hadoop/hdfs-site.xml

<configuration>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.name.dir</name>

<value>file:///home/**your\_user\_name**/hadoop/name\_dir</value>

</property>

<property>

<name>dfs.data.dir</name>

<value>file:///home/**your\_user\_name**/hadoop/data\_dir</value>

</property>

</configuration>

**#Setup passphraseless/passwordlessssh**ssh-keygen -t dsa -P '' -f ~/.ssh/id\_dsa cat ~/.ssh/id\_dsa.pub >> ~/.ssh/authorized\_keys export HADOOP\\_PREFIX=/home/**your\_user\_name**/hadoopsshlocalhost

# type**exit** in the terminal to close the ssh connection (very important) Exit

**# The following instructions are to run a MapReduce job locally.**

#Format the filesystem:( **Do it only once** )

~/hadoop/bin/hdfsnamenode -format

#Start NameNode daemon and DataNode daemon:

~/hadoop/sbin/start-dfs.sh

#Browse the web interface for the NameNode; by default it is available at: http://localhost:50070/

#Make the HDFS directories required to execute MapReduce jobs:

~/hadoop/bin/hdfsdfs -mkdir /user

~/hadoop/bin/hdfsdfs -mkdir /user/**your\_user\_name**

#Copy the sample files (from ~/hadoop/etc/hadoop) into the distributed filesystemfolder(input)

~/hadoop/bin/hdfsdfs -put ~/hadoop/etc/hadoop input

#Run the example map-reduce job

~/hadoop/bin/hadoop jar ~/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.1.jar grep input output 'us[a-z.]+'

#View the output files on the distributed filesystem

~/hadoop/bin/hdfsdfs -cat output/\*

#Copy the output files from the distributed filesystem to the local filesystem and examine them:

~/hadoop/bin/hdfsdfs -get output output

#ignore warnings (if any) cat output/\*

# remove local output folder rm -r output

# remove distributed folders (input & output) ~/hadoop/bin/hdfsdfs -rm -r input output #When you’re done, stop the daemons with

~/hadoop/sbin/stop-dfs.sh

**Conclusion:** In this way the Hadoop was installed & configured on Ubuntu for BigData.

# **b)Multiple Node**

## Install a Multi Node Hadoop Cluster on Ubuntu 14.04

You would need minimum of 2 ubuntu machines or virtual images to complete a multi-node installation.  If you want to just try out a single node cluster, follow this article on [Installing Hadoop on Ubuntu 14.04.](https://chawlasumit.wordpress.com/2014/06/15/installing-hadoop-on-ubuntu-14-04/)

.

## Installing Java on Master and Slaves

$ sudo add-apt-repository ppa:webupd8team/java

$ sudo apt-get update

$ sudo apt-get install oracle-java7-installer

# Updata Java runtime

$ sudo update-java-alternatives -s java-7-oracle

## Disable IPv6

As of now Hadoop does not support IPv6, and is tested to work only on IPv4 networks.   If you are using IPv6, you need to switch Hadoop host machines to use IPv4. [The Hadoop Wiki](http://wiki.apache.org/hadoop/HadoopIPv6) link provides a one liner command to disable the IPv6.  If you are not using IPv6, skip this step:

sudo**sed** -i '**s**/net.ipv6.bindv6only\ =\ 1/net.ipv6.bindv6only\ =\ 0/' \ /etc/sysctl.d/bindv6only.conf &&sudo invoke-rc**.**dprocps restart

|  |
| --- |
|  |

## Setting up a Hadoop User

Hadoop talks to other nodes in the cluster using no-password ssh.   By having Hadoop run under a specific user context, it will be easy to distribute the ssh keys around in the Hadoop cluster.  Lets’s create a user **hadoopuser**on **master** as well as **slave** nodes.

# Createhadoopgroup

$ sudoaddgrouphadoopgroup

# Createhadoopuser user

$ sudoadduser —ingrouphadoopgrouphadoopuser

Our next step will be to generate a ssh key for password-less login between master and slave nodes.  Run the following commands only on **master** node.  Run the last two commands for each slave node.  Password less ssh should be working before you can proceed with further steps.

|  |
| --- |
| # Login as hadoopuser  $ su - hadoopuser  #Generate a ssh key for the user  $ ssh-keygen -t rsa -P ""  #Authorize the key to enable password less ssh  $ cat /home/hadoopuser/**.**ssh/id\_rsa.pub >> /home/hadoopuser/**.**ssh/authorized\_keys $ chmod 600 authorized\_keys  #Copy this key to slave-1 to enable password less ssh  $ ssh-copy-id -i ~/**.**ssh/id\_rsa.pub slave-1  #Make sure you can do a password less ssh using following command. $ ssh slave-1 |

## Download and Install Hadoop binaries on Master and Slave nodes

Pick the best mirror site to download the binaries from [Apache Hadoop,](http://www.apache.org/dyn/closer.cgi/hadoop/core/) and download the stable/hadoop-2.6.0.tar.gz for your installation.  **Do this step on master and every slave node.**  You can download the file once and the distribute to each slave node using scp command.

$ **cd** /home/hadoopuser

$ wget http://www.webhostingjams.com/mirror/apache/hadoop/core/stable/hadoop-2.2.0.tar.gz $ tar xvf hadoop-2**.**2**.**0**.**tar**.**gz

$ mv hadoop-2**.**2**.**0 hadoop

## Setup Hadoop Environment on Master and Slave Nodes

Copy and paste following lines into your .bashrc file under /home/hadoopuser. **Do this step on master and every slave node.**

# Set HADOOP\_HOME

**export** HADOOP\_HOME=/home/hduser/hadoop

# Set JAVA\_HOME

**export** JAVA\_HOME=/usr/lib/jvm/java-7-oracle

# AddHadoop bin and sbin directory to PATH

**export** PATH=$PATH:$HADOOP\_HOME/bin;$HADOOP\_HOME/sbin

## Update hadoop-env.sh on Master and Slave Nodes

Update JAVA\_HOME in /home/hadoopuser/hadoop/etc/hadoop/hadoop\_env.sh to following. **Do this step on master and every slave node.**

**export** JAVA\_HOME=/usr/lib/jvm/java-7-oracle

## Common Terminologies

Before we start getting into configuration details, lets discuss some of the basic terminologies used in Hadoop.

* **Hadoop Distributed File System**: A distributed file system that provides high-throughput access to application data. A HDFS cluster primarily consists of a NameNode that manages the file system metadata and DataNodes that store the actual data. If you compare HDFS to a traditional storage structures ( e.g. FAT, NTFS), then NameNode is analogous to a Directory Node structure, and DataNode is analogous to actual file storage blocks.
* **Hadoop YARN**: A framework for job scheduling and cluster resource management.
* **HadoopMapReduce**: A YARN-based system for parallel processing of large data sets.

## Update Configuration Files

Add/update core-site.xml on **Master and Slave nodes** with following options.  Master and slave nodes should all be using the same value for this property **fs.defaultFS,** and should be pointing to master node only.

|  |
| --- |
| **/home/hadoopuser/hadoop/etc/hadoop/core-site.xml** [**(Other Options)**](http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/core-default.xml) |
| <property>    <name>hadoop.tmp.dir</name>    <value>/home/hadoopuser/tmp</value>    <description>Temporary Directory.</description>  </property>    <property>    <name>fs.defaultFS</name>    <value>hdfs://master:54310</value>    <description>Use HDFS as file storage engine</description></property> |

Add yarn-site.xml on **Master and Slave Nodes**. This file is required for a Node to work as a Yarn Node. Master and slave nodes should all be using the same value for the following properties**,** and should be pointing to master node only.

|  |
| --- |
| **/home/hadoopuser/hadoop/etc/hadoop/yarn-site.xml** |
| <property>  <name>yarn.nodemanager.aux-services</name>  <value>mapreduce\_shuffle</value>  </property>  <property>  <name>yarn.resourcemanager.scheduler.address</name>  <value>master:8030</value>  </property>  <property>  <name>yarn.resourcemanager.address</name>  <value>master:8032</value>  </property>  <property>  <name>yarn.resourcemanager.webapp.address</name>  <value>master:8088</value>  </property>  <property>  <name>yarn.resourcemanager.resource-tracker.address</name>  <value>master:8031</value>  </property>  <property>  <name>yarn.resourcemanager.admin.address</name>  <value>master:8033</value>  </property> |

Add/update **slaves** file on Master node only. Add just name, or ip addresses of master and all slave node. If file has an entry for localhost, you can remove that. This file is just helper file that are used by hadoop scripts to start appropriate services on master and slave nodes.

|  |
| --- |
| **/home/hadoopuser/hadoop/etc/hadoop/slave** |
| master slave-1 slave-2 |

## **Format the Namenode**

Before starting the cluster, we need to format the Namenode. Use the following command only on **master node**:

$ hdfsnamenode –format

|  |
| --- |
|  |

## **Start the Distributed Format System**

Run the following on **master node** command to start the DFS.

$ **.**/home/hadoopuser/hadoop/sbin/start-dfs.sh

You should observe the output to ascertain that it tries to start datanode on slave nodes one by one. To validate the success, run following command on master nodes, and slave node.

$ su - hadoopuser

$ jps

The output of this command should list ***NameNode*,*SecondaryNameNode, DataNode*** on **master** node, and ***DataNode***on all slave nodes. If you don’t see the expected output, review the log files listed in Troubleshooting section.

## **Start the Yarn MapReduce Job tracker**

Run the following command to start the Yarn mapreduce framework.

$ **.**/home/hadoopuser/hadoop/sbin/start-yarn.sh

To validate the success, run **jps** command again on master nodes, and slave node.The output of this command should list ***NodeManager, ResourceManager*** on **master** node, and ***NodeManager,***on all **slave** nodes. If you don’t see the expected output, review the log files listed in Troubleshooting section.

## **Review Yarn Web console**

If all the services started successfully on all nodes, then you should see all of your nodes listed under Yarn nodes. You can hit the following url on your browser and verify

that:http://master:8088/cluster/nodes

## **Lets’s execute a MapReduce example now**

You should be all set to run a MapReduce example now. Run the following command

$ hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.2.0.jar pi 30 100

Once the job is submitted you can validate that its running on the cluster by accessing following url. http://master:8088/cluster/apps

**Conclusion:** Thus we have tested successfully multimode cluster.

##### /home/hadoopuser/hadoop/

**ASSIGNMENTNO.2**

##### ProblemStatement:

**Aim:** Design a distributed application using MapReduce(Using Java) which processes a log file of a system. List out the users who have logged for maximum period on the system. Use simple log file from the Internet and process it using a pseudo distribution mode on Hadoop platform.

##### Tools/Environment:

Ubuntu

RelatedTheory:

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. 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/value pairs).

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.

MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.

**Mapstage :** 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.

**Inserting Data into HDFS:**

•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).

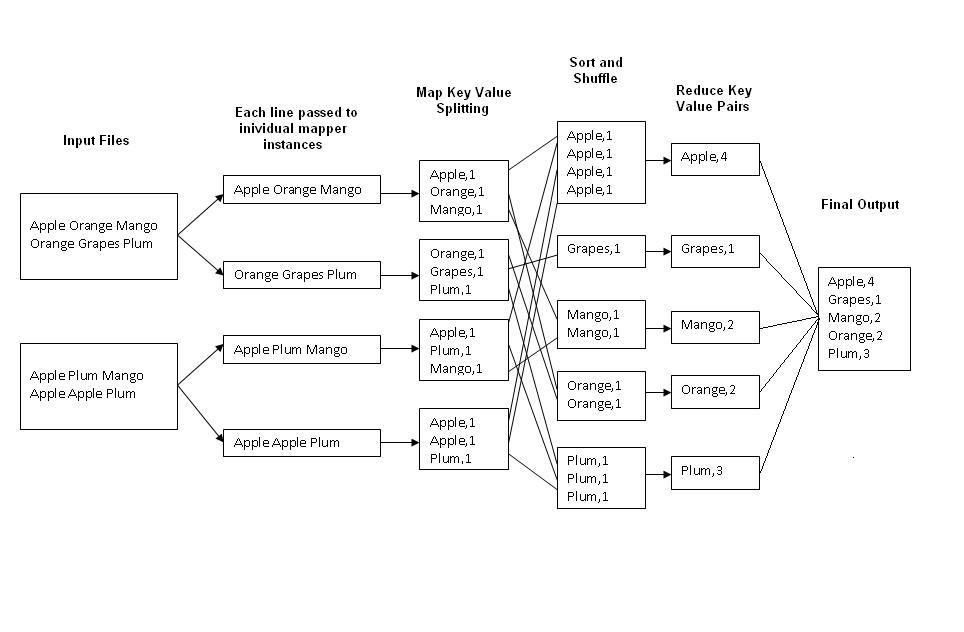


Fig.1 : An Example Program to Understand working of MapReduce Program.

**Steps for Compilation & Execution of Program:**

Su hadoopuser

#sudomkdiranalyzelogs

ls

#sudochmod -R 777 analyzelogs/

cd

ls cd .. (to move to home directory)

pwd

ls

cd

pwd

#sudochown -R hadoop1 analyzelogs/

cd

ls

#cd analyzelogs/

ls cd ..

# Copy the Files (Mapper.java,Reduce.java,Driver.java to Analyzelogs Folder)

# #sudocp /home/priyanka/Desktop/assignment3/\* -/analyzelogs/

(Convert access\_log\_short.txt into access\_log\_short.csv)

Start HADOOP

#start-dfs.sh

#start-yarn.sh

#jps

cd

cdanalyzelogs

ls pwd

ls

#ls -ltr

#ls -al

#sudochmod +r \*.\*

pwd

#export CLASSPATH="$HADOOP\_HOME/share/hadoop/mapreduce/hadoop-mapreduceclient-core-2.9.0.jar:$HADOOP\_HOME/share/hadoop/mapreduce/hadoop-mapreduce-clientcommon-2.9.0.jar:$HADOOP\_HOME/share/hadoop/common/hadoop-common-2.9.0.jar**:~/analyzelogs/SalesCountry**/\*:$HADOOP\_HOME/lib/\*"

(This should be PWD)

# **Compile Java Files**

# javac -d . SalesMapper.java SalesCountryReducer.java SalesCountryDriver.java ls

#cd SalesCountry/

ls cd ..

#sudogedit Manifest.txt

Main-class:SalesCountry.SalesCountryDriver

(Press enter)

#jar -cfm analyzelogs.jar Manifest.txt SalesCountry/\*.class ls cd

#cd analyzelogs/

# **Create Directory on Hadoop**

#sudo mkdir ~/input2000

ls

pwd

#sudocp access\_log\_short.csv ~/input2000/

# $HADOOP\_HOME/bin/hdfsdfs -put ~/input2000 /

# $HADOOP\_HOME/bin/hadoop jar analyzelogs.jar /input2000 /output2000

# $HADOOP\_HOME/bin/hdfsdfs -cat /output2000/part-00000

# stop-all.sh

# jps

For GUI

Go to browser(localhost:50070)

Go to utilities(browse directory)

**Conclusion:** Thus we have learnt how to design a distributed application using MapReduce and process a log file of a system.

**etc/hadoop/core-site.xml** [**(Other Options)**](http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/core-default.xml)

**ASSIGNMENT NO.3**

##### Problem Statement:

**Aim:** Write an application using HiveQL for flight information system which will include

**a.** Creating, Dropping, and altering Database tables.

**b.** Creating an external Hive table.

**c.** Load table with data, insert new values and field in the table, Join tables with Hive

**d.** Create index on Flight Information Table

**e.** Find the average departure delay per day in 2008.

##### Tools/Environment:

Ubuntu

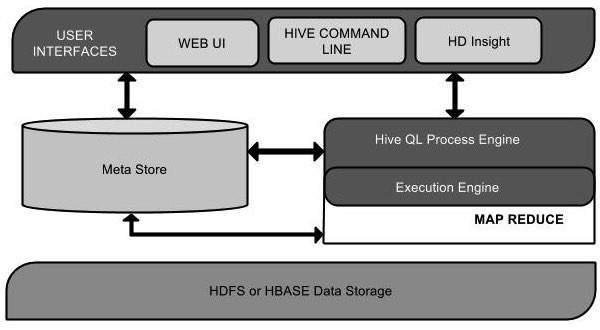
RelatedTheory:

**Hive:**

Apache Hive is a data warehouse system developed by Facebook to process a huge amount of structure data in Hadoop. We know that to process the data using Hadoop, we need to right complex map-reduce functions which is not an easy task for most of the developers. Hive makes this work very easy for us.

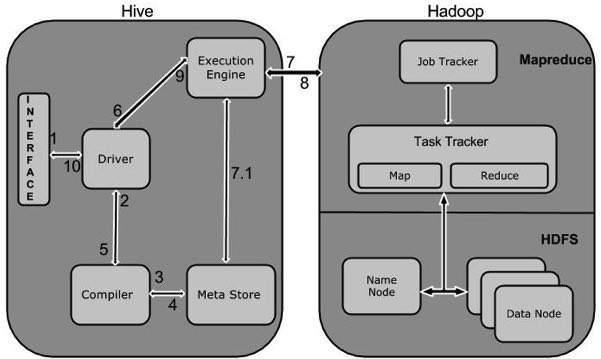
It uses a scripting language called HiveQL which is almost similar to the SQL. So now, we just have to write SQL-like commands and at the backend of Hive will automatically convert them into the map-reduce jobs.

**Hive architecture:**



Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping. HiveQL is similar to SQL for querying on schema info on the Metastore.  Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce. Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.

**Working of Hive:**



**Data Types in Apache Hive**

Hive data types are divided into the following 5 different categories:

1. Numeric Type: TINYINT, SMALLINT, INT, BIGINT
2. Date/Time Types: TIMESTAMP, DATE, INTERVAL
3. String Types: STRING, VARCHAR, CHAR
4. Complex Types: STRUCT, MAP, UNION, ARRAY
5. Misc Types: BOOLEAN, BINARY