MAHARSHI DAYANAND UNIVERSITY

DELHI GLOBAL INSTITUTE OF TECHNOLOGY

BIG DATA ANALYTICS LAB

PRACTICAL FILE

SUBJECT CODE: LC-CSE-421G

Submitted By : Bazgha Razi

Registration No. : 191380214

INDEX

PRACTICAL NO – 1

Aim: Installation of Single Node Hadoop Cluster on Ubuntu

THEORY:

Apache Hadoop 3.1 have noticeable improvements any many bug fixes over the previous stable 3.0 releases. This version has many improvements in HDFS and MapReduce. This how-to guide will help you to setup Hadoop 3.1.0 Single-Node Cluster on CentOS/RHEL 7/6/5, Ubuntu 18.04, 17.10, 16.04 & 14
.04, Debian 9/8/7 and LinuxMint Systems. This article has been tested with Ubuntu 18.04 LTS.

1. Prerequisites

Java is the primary requirement for running Hadoop on any system, So make sure you have Java installed on your system using the following command. If you don't have Java installed on your system, use one of the following links to install it first. Hadoop supports only JAVA 8 If already any other version is present then uninstall the following using these commands.

sudo apt-get purge openjdk-* icedtea-* icedtea6-*

OR

sudo apt remove openjdk-8-jdk

- **Step 1.1 – Install Oracle Java 8 on Ubuntu**

  You need to enable additional repository to your system to install Java 8 on Ubuntu VPS. After that install Oracle Java 8 on an Ubuntu system using apt-get.This repository contains a package named oracle-java8-installer, Which is notan actual Java package. Instead of that, this package contains a script toinstall Java on Ubuntu.Run below commands to install Java 8 on Ubuntu and LinuxMint.

  sudo add-apt-repository

  ppa:webupd8team/java sudo apt-get

  sudo apt-get install oracle-java8-installer

  OR

  sudo apt install openjdk-8-jre-headless

  sudo apt install openjdk-8-jdk

- **Step 1.2 – Verify Java Inatallation**

  The apt repository also provides package oracle-java8-set-default to set Java 8 as your default Java version. This package will be installed along with Java installation. To make sure run below command.

  sudo apt-get install oracle-java8-set-default

  After successfully installing Oracle Java 8 using the above steps, Let's verify the installed version using the following command.

  java -version

  java version "1.8.0_201"

  Java(TM) SE Runtime Environment (build 1.8.0_201-b09)

  Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode)

- **Step 1.3 – Setup JAVA_HOME and JRE_HOME Variable**

  Add the java path to JAVA_HOME variable in .bashrc file. Go to your home directory and in the folder option click on show hidden files. After that a .bashrc file will be present, open the file and add the following line at the end.

  NOTE- Path of the java will be your pc path on which java is been installed. export JAVA_HOME=/usr/lib/jvm/java-8-oracle

  NOTE- After doing all changes and saving the file run the following command to make changes through the .bashrc file.

  source ~/.bashrc

  All done, you have successfully installed Java 8 on a Linux system.

2. Create Hadoop User

We recommend creating a normal (nor root) account for Hadoop working. To create an account using the following command.

adduser hadoop

passwd hadoop

Set up a new user for Hadoop working separately other than the normal users.

NOTE- Its compulsory to create a sperate user with username hadoop otherwise it may give you path file issues later.

Also run these commands from an admin privileged user present on the machine. sudo adduser Hadoop sudo

Command – sudo adduser hadoop sudo If you have already created the user and want to give sudo/root privileges to it then run the following command.

sudo usermod -a -G sudo hadoop

Otherwise you can directly edit the permission lines in sudoers file. Go to the root access by running

sudo -I or su- <username>

Type the following command and add the below line to the file.

visudo

Add following lines to the file.

hadoop ALL=(ALL:ALL) ALL

After creating the account, it also required to set up key-based ssh to its own account. To do this use execute following commands.

su - hadoop

ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa

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

chmod 0600 ~/.ssh/authorized_keys

Let's verify key based login. Below command should not ask for the password but the first time it will prompt for adding RSA to the list of known hosts.

ssh localhost

exit

Disable all firewall restriction

sudo ufw disable

If above command doesn't work then go with.
service iptables stop
OR
sudo chkconfig iptables off
Sometimes it's better to manage firewall using a third party software. Ex. yast

3. Download Hadoop 3.1 Archive

In this step, download hadoop 3.1 source archive file using below command. You can also select alternate download mirror for increasing download speed.

```
cd ~
wget http://www-eu.apache.org/dist/hadoop/common/hadoop-3.1.0/hadoop-3.1.0.tar.gz
tar xzf hadoop-3.1.0.tar.gz
mv hadoop-3.1.0 hadoop
```

4. Setup Hadoop Pseudo-Distributed Mode

Setup Hadoop Environment Variables

First, we need to set environment variable uses by Hadoop. Edit ~/.bashrc file and append following values at end of file.

```
export HADOOP_HOME=/home/hadoop/hadoop export
HADOOP_INSTALL=$HADOOP_HOME export
HADOOP_MAPRED_HOME=$HADOOP_HOME export
HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
```

Now apply the changes in the current running environment source ~/.bashrc

Now edit $HADOOP_HOME/etc/hadoop/hadoop-env.sh file and set JAVA_HOME environment variable. Change the JAVA path as per install on your system. This path may vary as per your operating system version and installation source. So make sure you are using correct path. export JAVA_HOME=/usr/lib/jvm/java-8-oracle

Setup Hadoop Configuration Files

Hadoop has many of configuration files, which need to configure as per requirements of your Hadoop infrastructure. Let's start with the configuration with basic Hadoop single node cluster setup. first, navigate to below location

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

Edit core-site.xml

```
<configuration>
<property>
<name>fs.default.name</name>
```

<value>hdfs://localhost:9000</value>
</property>
</configuration>

Edit hdfs-site.xml
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.name.dir</name>
<value>file:///home/hadoop/hadoopdata/hdfs/namenode</value>
</property>
<property>
<name>dfs.data.dir</name>
<value>file:///home/hadoop/hadoopdata/hdfs/datanode</value>
</property>
</configuration>

Edit mapred-site.xml
<configuration>
<property>
<name>mapreduce.framework.name </name>
<value>yarn </value>
</property>
</configuration>

Edit yarn-site.xml
<configuration>
<property>
<name>yarn.nodemanager.aux-services </name>
<value>mapreduce_shuffle </value>
</property>
</configuration>

Format Namenode

Now format the namenode using the following command, make sure that Storage directory is hdfs namenode -format

Sample output:
WARNING: /home/hadoop/hadoop/logs does not exist. Creating.
2018-05-02 17:52:09,678 INFO namenode.NameNode: STARTUP_MSG:

/**
STARTUP_MSG: Starting NameNode STARTUP_MSG: host =
localhost/127.0.1.1
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 3.1.0
...
...
...
2018-05-02 17:52:13,717 INFO common.Storage: Storage directory
/home/hadoop/hadoopdata/hdfs/namenode has been successfully formatted.
2018-05-02 17:52:13,806 INFO namenode.FSImageFormatProtobuf: Saving image file
/home/hadoop/hadoopdata/hdfs/namenode/current/fsimage.ckpt_0000000000000000000 using
no
compression
2018-05-02 17:52:14,161 INFO namenode.FSImageFormatProtobuf: Image file
/home/hadoop/hadoopdata/hdfs/namenode/current/fsimage.ckpt_0000000000000000000 of size
391 bytes saved in 0 seconds .
2018-05-02 17:52:14,224 INFO namenode.NNStorageRetentionManager: Going to retain
1 images with txid >= 0
2018-05-02 17:52:14,282 INFO namenode.NameNode: SHUTDOWN_MSG:
/**
SHUTDOWN_MSG: Shutting down NameNode at localhost/127.0.1.1
**/

5. Start Hadoop Cluster

Let's start your Hadoop cluster using the scripts provides by Hadoop. Just navigate to your $HADOOP_HOME/sbin directory and execute scripts one by one.

Cd $HADOOP_HOME/sbin/

Now run start-dfs.sh script.

./start-dfs.sh

Sample output:Starting namenodes on
[localhost] Starting datanodes
Starting secondary namenodes [localhost]
2018-05-02 18:00:32,565 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable

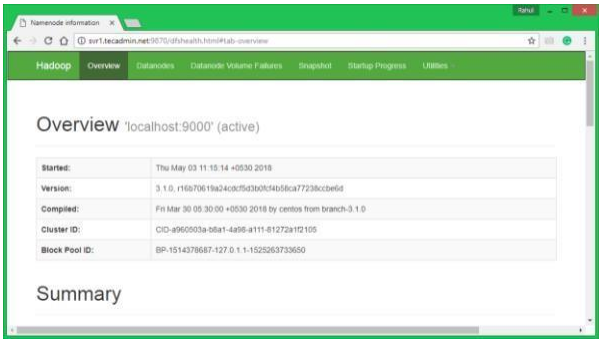
Now run start-yarn.sh script.

./start-yarn.sh

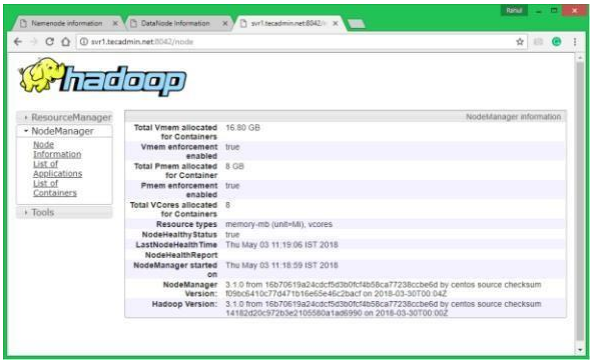
Sample output:Starting resourcemanager
Starting nodemanagers

6. Access Hadoop Services in Browser

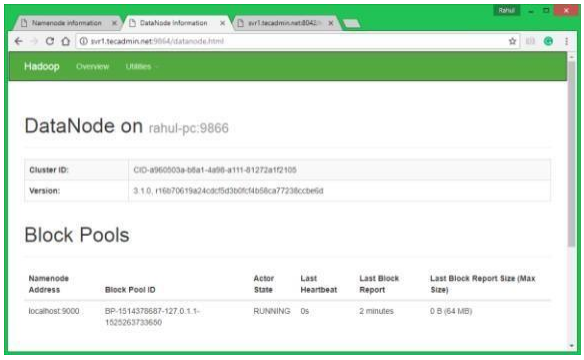
Hadoop NameNode started on port 9870 default. Access your server on port 9870 in your favorite web browser.
http://localhost:9870/



Now access port 8042 for getting the information about the cluster and all applications http:// localhost:8042/



Access port 9864 to get details about your Hadoop node.
http://localhost:9864/



7. Test Hadoop Single Node Setup

Make the HDFS directories required using following commands.

bin/hdfs dfs -mkdir /user

bin/hdfs dfs -mkdir /user/hadoop

Copy all files from local file system /var/log/httpd to hadoop distributed file system using below command

bin/hdfs dfs -put /var/log/apache2 logs

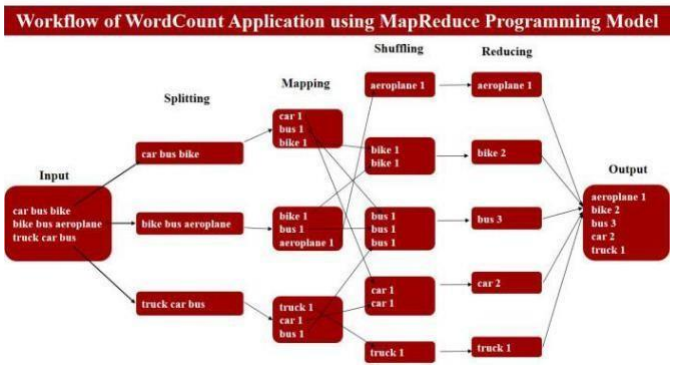
Browse Hadoop distributed file system by opening below URL in the browser. You willsee an apache2 folder in the list.

http://localhost:9870/explorer.html#/user/hadoop/logs/

PRACTICAL NO – 2

Aim: Hadoop Programming: Word Count MapReduce Program Using Eclipse

THEORY:



Steps to run WordCount Application in Eclipse

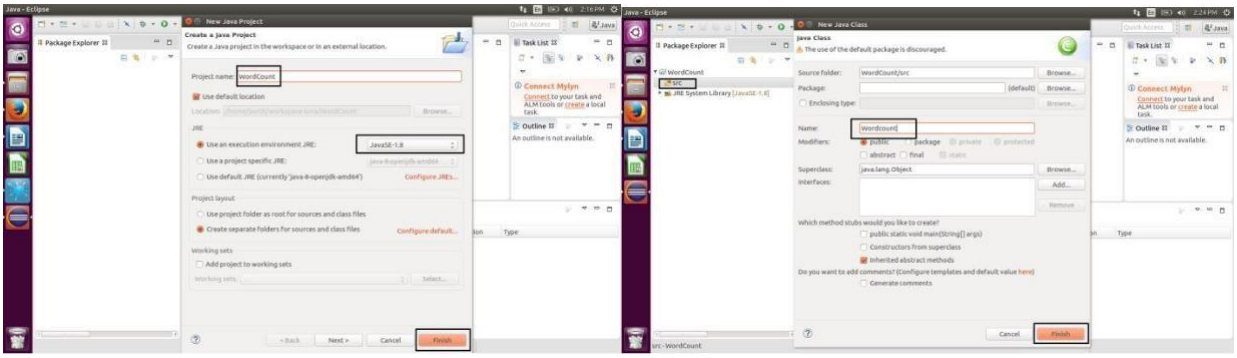
Step-1

Download eclipse if you don't have. 64 bit Linux os 32 bit Linux os

Step-2

Open Eclipse and Make Java Project.

In eclipse Click on File menu-> new -> Java Project. Write there your project name. Here is WordCount. Make sure Java version must be 1.6 and above. Click on Finish.



Step-3

Make Java class File and write a code.

Click on WordCount project. There will be 'src' folder. Right click on 'src' folder -> New -> Class. Write Class file name. Here is Wordcount. Click on Finish.

Copy and Paste below code in Wordcount.java. Save it.

You will get lots of error but don't panic. It is because of requirement of external library of hadoop which is required to run mapreduce program.

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
```

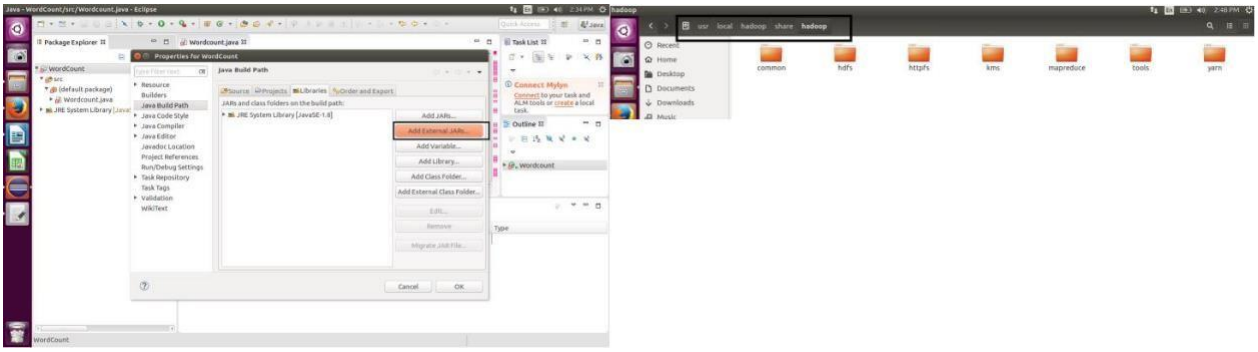
```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class Wordcount {
 public static class TokenizerMapper
 extends Mapper{
 private final static IntWritable one = new
 IntWritable(1); private Text word = new Text();
 public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
 StringTokenizer itr = new StringTokenizer(value.toString());
 while (itr.hasMoreTokens()) {
 word.set(itr.nextToken());
 context.write(word, one);
 }
 }
 }
 public static class IntSumReducer
 extends Reducer {
 private IntWritable result = new IntWritable();
 public void reduce(Text key, Iterable values,Context context) throws
IOException, InterruptedException {
 int sum = 0;
 for (IntWritable val : values) {
 sum += val.get();
 }
 result.set(sum);
 context.write(key, result);
 }
 }
 public static void main(String[] args) throws Exception
 { Configuration conf = new Configuration();
 Job job = Job.getInstance(conf, "word count");
 job.setJarByClass(WordCount.class);
 job.setMapperClass(TokenizerMapper.class);
 job.setCombinerClass(IntSumReducer.class);
 job.setReducerClass(IntSumReducer.class);
 job.setOutputKeyClass(Text.class);
 job.setOutputValueClass(IntWritable.class);
 FileInputFormat.addInputPath(job, new Path(args[0]));
 FileOutputFormat.setOutputPath(job, new Path(args[1]));
 System.exit(job.waitForCompletion(true) ? 0 : 1);
```

```
 }
}
```

Step-4

Add external libraries from hadoop.

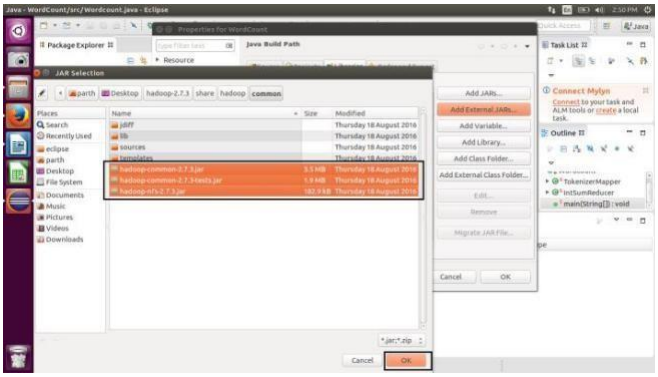
Right click on WordCount Project -> Build Path -> Configure Build Path -> Click on Libraries -> click on ‘Add External Jars..’ button.



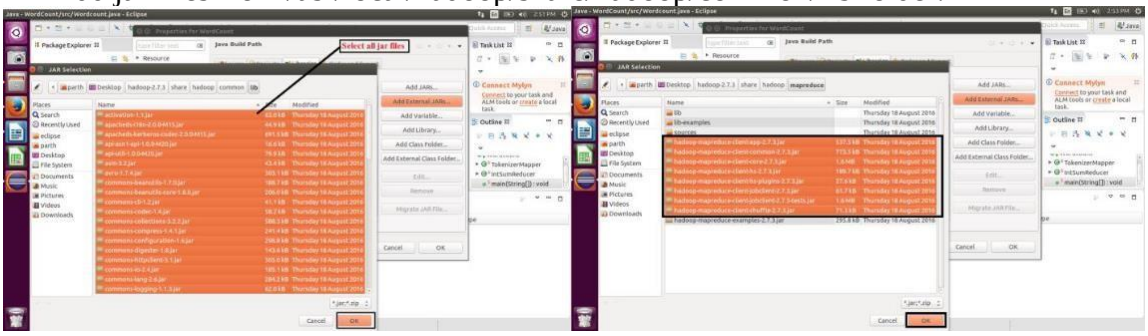
Select below files from hadoop folder.

In my case:- /usr/local/hadoop/share/hadoop

Add jar files from /usr/local/hadoop/share/hadoop/common folder.

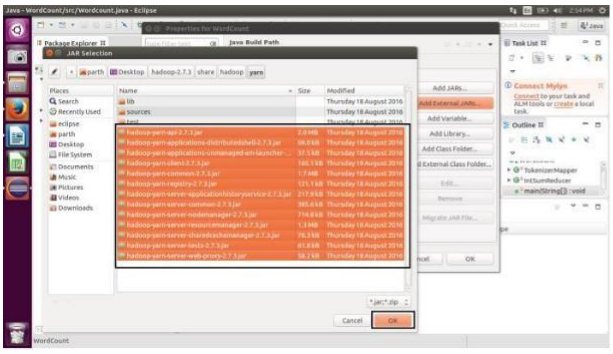


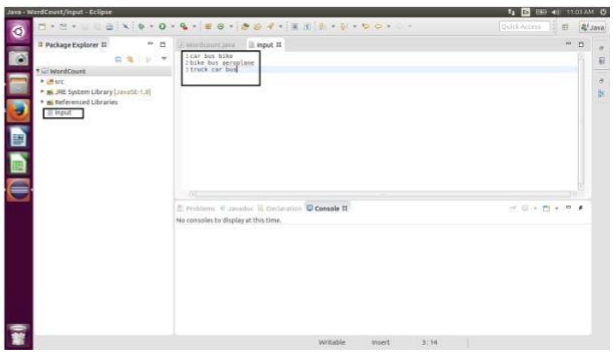
Add jar files from /usr/local/hadoop/share/hadoop/common/lib folder.



Add jar files from /usr/local/hadoop/share/hadoop/mapreduce folder (Don’t need to add hadoop-mapreduce-examples-2.7.3.jar)

Add jar files from /usr/local/hadoop/share/hadoop/yarn folder.





Click on ok. Now you can see, all error in code is gone.

Step 5

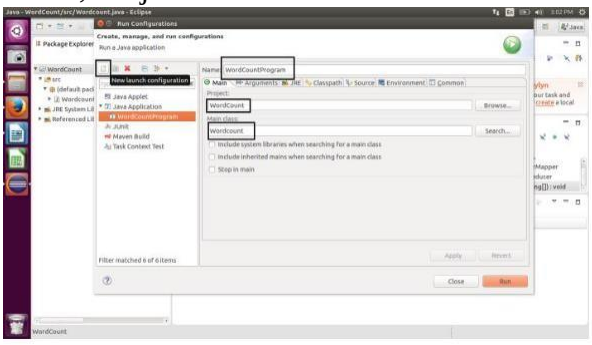
Running Mapreduce Code.

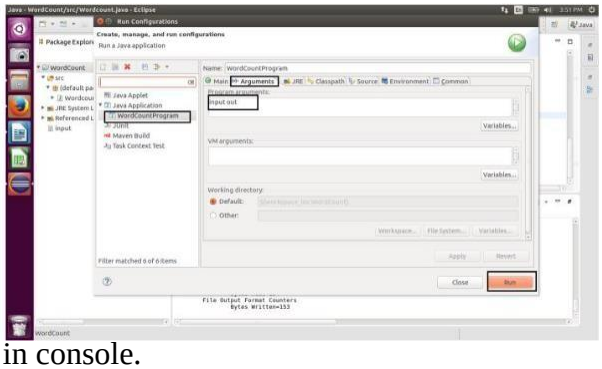
Make input file for WordCount Project.

Right Click on WordCount project-> new -> File. Write File name and click on ok. You can copy and paste below contains into your input file.

car bus bike
bike bus aeroplane
truck car bus

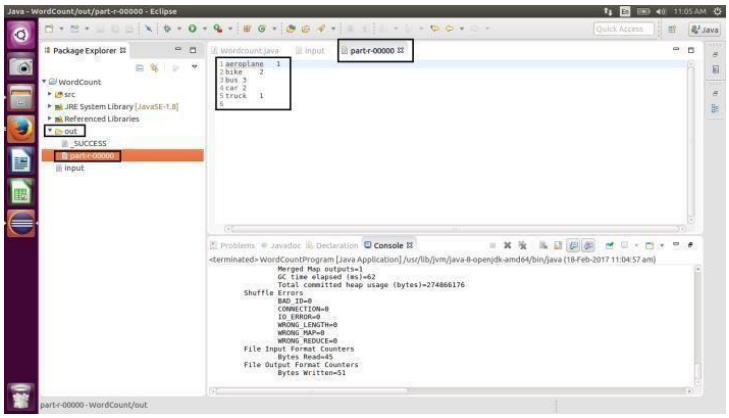
Right click on WordCount Project -> click on Run As. -> click on Run Configuration…
Make new configuration by clicking on 'new launch configuration'. Set Configuration Name, Project Name and Class file name.





Output of WordCount Application and output logs in console.

Refresh WordCount Project. Right Click on project -> click on Refresh. You can find 'out' directory in project explorer. Open 'out' directory. There will be 'part-r-00000' file. Double click to open it.

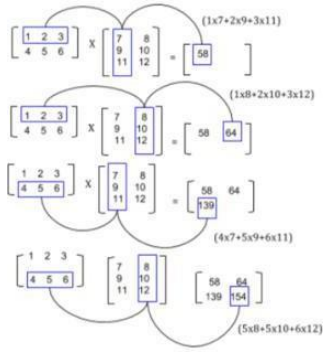


PRACTICAL NO – 3

Aim: Implementing Matrix Multiplication Using One Map-Reduce Step.

THEORY:

In mathematics, matrix multiplication or the matrix product is a binary operation that produces a matrix from two matrices. In more detail, if A is an n × m matrix and B is an m × p matrix, their matrix product AB is an n × p matrix, in which the m entries across a row of A are multiplied with the m entries down a column of B and summed to produce an entry of AB. When two linear transformations are represented by matrices, then the matrix product represents the composition of the two transformations.



Algorithm for Map Function:

for each element mij of M do

produce (key,value) pairs as ((i,k), (M,j,mij), for k=1,2,3,.. upto the number of columns of N

for each element njk of N do

produce (key,value) pairs as ((i,k),(N,j,Njk), for i = 1,2,3,.. Upto the number of rows of M.

return Set of (key,value) pairs that each key (i,k), has list with values (M,j,mij) and (N, j,njk) for all possible values of j.

Algorithm for Reduce Function:

for each key (i,k) do

sort values begin with M by j in listM

sort values begin with N by j in listN

multiply mij and njk for jth value of each list

sum up mij x njk return (i,k), Σj=1 mij x njk

Step 1. Download the hadoop jar files with these links.

Download Hadoop Common Jar files: https://goo.gl/G4MyHp $

wget https://goo.gl/G4MyHp -O hadoop-common-2.2.0.jar

Download Hadoop Mapreduce Jar File: https://goo.gl/KT8yfB

$ wget https://goo.gl/KT8yfB -O hadoop-mapreduce-client-core-2.7.1.jar

Step 2. Creating Mapper file for Matrix Multiplication.

```
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
public class Map
 extends org.apache.hadoop.mapreduce.Mapper<LongWritable, Text, Text, Text> {
    @Override
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        Configuration conf = context.getConfiguration();int
        m = Integer.parseInt(conf.get("m")); int p =
        Integer.parseInt(conf.get("p"));
        String line = value.toString();
        // (M, i, j, Mij);
```

```
            String[] indicesAndValue = line.split(",");
            Text outputKey = new Text();
            Text outputValue = new Text();
            if (indicesAndValue[0].equals("M")) {
                for (int k = 0; k < p; k++) {
                    outputKey.set(indicesAndValue[1] + "," + k);
                    // outputKey.set(i,k);
                    outputValue.set(indicesAndValue[0] + "," + indicesAndValue[2]
                            + "," + indicesAndValue[3]);
                    // outputValue.set(M,j,Mij);
                    context.write(outputKey, outputValue);
                }
            } else {
                // (N, j, k, Njk);
                for (int i = 0; i < m; i++) {
                    outputKey.set(i + "," + indicesAndValue[2]);
                    outputValue.set("N," + indicesAndValue[1] + ","
                            + indicesAndValue[3]);
                    context.write(outputKey, outputValue);
                }
            }
        }
    }
```

Step 3. Creating Reducer.java file for Matrix Multiplication.

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
import java.util.HashMap;
public class Reduce
  extends org.apache.hadoop.mapreduce.Reducer<Text, Text, Text, Text>
      { @Override
      public void reduce(Text key, Iterable<Text> values, Context context)
                  throws IOException, InterruptedException {
            String[] value;
            //key=(i,k),
            //Values = [(M/N,j,V/W),..]
            HashMap<Integer, Float> hashA = new  HashMap<Integer, Float>();
            HashMap<Integer, Float> hashB = new HashMap<Integer,Float>();
            for (Text val : values) {
                  value = val.toString().split(",");
                  if (value[0].equals("M")) {
                        hashA.put(Integer.parseInt(value[1]),
            Float.parseFloat(value[2])); } else {
                        hashB.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));
                  }
            }
```

```
            int n = Integer.parseInt(context.getConfiguration().get("n"));
            float result = 0.0f;
            float m_ij;
            float n_jk;
            for (int j = 0; j < n; j++) {
                m_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;
                n_jk = hashB.containsKey(j) ? hashB.get(j) :0.0f;
                result += m_ij * n_jk;
            }
            if (result != 0.0f) {
                context.write(null,
                        new Text(key.toString() + "," + Float.toString(result)));
            }
        }
}
```

Step 4. Creating MatrixMultiply.java file for

```
import  org.apache.hadoop.conf.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

```
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class MatrixMultiply {
    public static void main(String[] args) throws Exception
        { if (args.length != 2) {
            System.err.println("Usage: MatrixMultiply <in_dir>
            <out_dir>"); System.exit(2);
        }
        Configuration conf = new Configuration();
        conf.set("m", "1000");
        conf.set("n", "100");
        conf.set("p", "1000");
        @SuppressWarnings("deprecation")
                Job job = new Job(conf, "MatrixMultiply");
        job.setJarByClass(MatrixMultiply.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new
        Path(args[1])); job.waitForCompletion(true);
```

```
    }

}
```

Step 5. Compiling the program in particular folder named as operation/

```
$ javac -cp hadoop-common-2.2.0.jar:hadoop-mapreduce-client-core-2.7.1.jar:operation/:. -d operation/ Map.java

$ javac -cp hadoop-common-2.2.0.jar:hadoop-mapreduce-client-core-2.7.1.jar:operation/:. -d operation/ Reduce.java

$ javac -cp hadoop-common-2.2.0.jar:hadoop-mapreduce-client-core-2.7.1.jar:operation/:. -d operation/ MatrixMultiply.java
```

Step 6. Let's retrieve the directory after compilation.

```
$ ls -R operation/

operation/:

www

operation/www:

ehadoopinfo

operation/www/ehadoopinfo:

com

operation/www/ehadoopinfo/com:

Map.class  MatrixMultiply.class  Reduce.class
```

Step 7. Creating Jar file for the Matrix Multiplication.

```
$ jar -cvf MatrixMultiply.jar -C operation/ .

added manifest

adding: www/(in = 0) (out= 0)(stored 0%)

adding: www/ehadoopinfo/(in = 0) (out= 0)(stored 0%)

adding: www/ehadoopinfo/com/(in = 0) (out= 0)(stored 0%)
```

adding: www/ehadoopinfo/com/Reduce.class(in = 2919) (out= 1271)(deflated 56%)

adding: www/ehadoopinfo/com/MatrixMultiply.class(in = 1815) (out= 932)(deflated 48%)

adding: www/ehadoopinfo/com/Map.class(in = 2353) (out= 993)(deflated 57%)

Step 8. Uploading the M, N file which contains the matrix multiplication data to HDFS.

$ cat M

M,0,0,1

M,0,1,2

M,1,0,3

M,1,1,4

$ cat N

N,0,0,5

N,0,1,6

N,1,0,7

N,1,1,8

$ hadoop fs -mkdir Matrix/

$ hadoop fs -copyFromLocal M Matrix/

$ hadoop fs -copyFromLocal N Matrix/

Step 9. Getting Output from part-r-00000 that was generated after the execution of the hadoop command.

$ hadoop fs -cat result/part-r-00000

0,0,19.0

0,1,22.0

1,0,43.0

1,1,50.0

PRACTICAL NO – 4

Aim: Implementing Relational Algorithm on Pig.

THEORY:

In this instructional post, we will explore and understand few important relational operators in Pig which is widely used in big data industry. Before we understand relational operators, let us see what Pig is.

Apache Pig, developed by Yahoo! helps in analyzing large datasets and spend less time in writing mapper and reducer programs. Pig enables users to write complex data analysis code without prior knowledge of Java. Pig's simple SQL-like scripting language is called Pig Latin and has its own Pig runtime environment where PigLatin programs are executed. For more details, I would suggest you to go through this blog.

Once you complete this blog, I would suggest you to get your hands dirty with a POC from this blog.

Below are two datasets that will be used in this post.

Employee_details.txt

This data set have 4 columns i.e.

Emp_id: unique id for each employee

Name: name of the employee

Salary: salary of an employee

Ratings: Rating of an employee.

```
[acadgild@localhost pig]$ cat employee_details.txt
101,Amitabh,20000,1
102,Shahrukh,10000,2
103,Akshay,11000,3
104,Anubhav,5000,4
105,Pawan,2500,5
106,Aamir,25000,1
107,Salman,17500,2
108,Ranbir,14000,3
109,Katrina,1000,4
110,Priyanka,2000,5
111,Tushar,500,1
112,Ajay,5000,2
113,Jubeen,1000,1
114,Madhuri,2000,2
```

Employee_expenses.txt

This data set have 2 columns i.e.

Emp_id: id of an employee

Expense: expenses made by an employee

```
[acadgild@localhost pig]$ cat employee_expenses.txt
101     200
102     100
110     400
114     200
119     200
105     100
101     100
104     300
102     400
```

As you have got the idea of datasets, let us proceed and perform some relational operations using this data.

NOTE: All the analysis is performed in local mode. To work in local mode, you need to start pig grunt shell using “pig -x local” command.

Relational Operators:

Load

To load the data either from local filesystem or Hadoop filesystem.

Syntax:

LOAD ‘path_of_data’ [USING function] [AS
schema]; Where;

path_of_data : file/directory name in single quotes.

USING : is the keyword.

function : If you choose to omit this, default load function PigStorage() is used.

AS : is the keyword

schema : schema of your data along with data type.

Eg:

The file named employee_details.txt is comma separated file and we are going to load it from local file system.

A = LOAD ‘/home/acadgild/pig/employee_details.txt’ USING PigStorage(‘,’) AS (id:int, name:chararray, salary:int, ratings:int);

```
grunt> A = LOAD '/home/acadgild/pig/employee_details.txt' USING PigStorage(',') AS (id:int, name:chararray, salary:int, ratings:int);
2016-11-14 03:22:26,524 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapreduce.job.counters.limit is deprecated. Ins
max
2016-11-14 03:22:26,524 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, u
2016-11-14 03:22:26,525 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.
grunt> describe A
A: {id: int,name: chararray,salary: int,ratings: int}
```

To check the result, you can use DUMP command.

```
2016-11-14 03:24:17,267 [mair
(101,Amitabh,20000,1)
(102,Shahrukh,10000,2)
(103,Akshay,11000,3)
(104,Anubhav,5000,4)
(105,Pawan,2500,5)
(106,Aamir,25000,1)
(107,Salman,17500,2)
(108,Ranbir,14000,3)
(109,Katrina,1000,4)
(110,Priyanka,2000,5)
(111,Tushar,500,1)
(112,Ajay,5000,2)
(113,Jubeen,1000,1)
(114,Madhuri,2000,2)
```

Similarly, you can load another data into another relation, say ‘B’

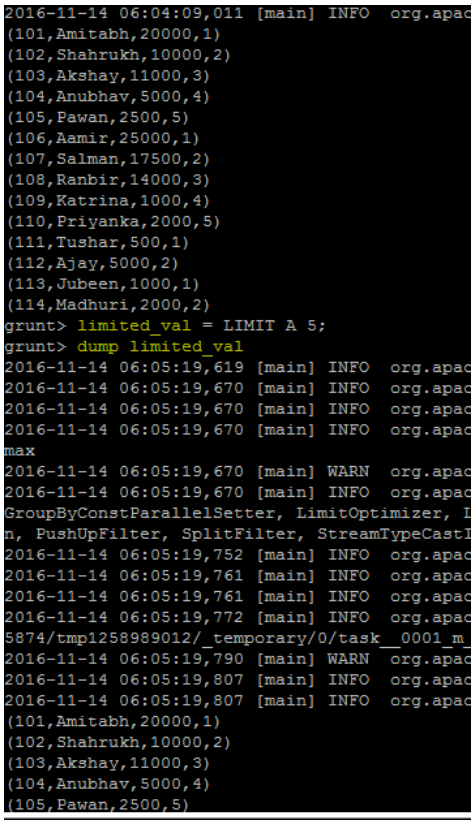
B = LOAD ‘/home/acadgild/pig/employee_expenses.txt’ USING PigStorage(‘\t’) AS (id:int, expenses:int);

As the fields in this file are tab separated, you need to use ‘\t’

NOTE: If you load this dataset in relation A, the earlier dataset will not be accessible.

```
grunt> B = LOAD '/home/acadgild/pig/employee_expenses.txt' USING PigStorage('\t') AS (id:int, expense:int);
2016-11-14 03:30:44,843 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapreduce.job.counters
max
2016-11-14 03:30:44,843 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum
2016-11-14 03:30:44,843 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is dep
```

Limit



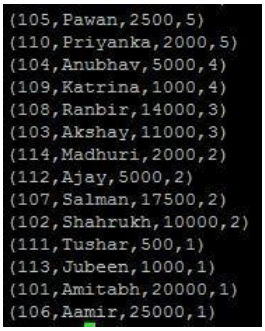
Used to limit the number of outputs to the desired number.
Syntax:
Alias = LIMIT alias n;
Where;
alias : name of the relation.
n : number of tuples to be displayed.
Ex:
We will be limiting the result of relation A (described above) to 5.
limited_val = LIMIT A 5;

NOTE: there is no guarantee which 5 tuples will be the output.

Order

Sorts a relation based on single or multiple fields.
Syntax:
alias = ORDER alias BY {field_name [ASC | DESC]
Where; alias : is the relation
ORDER : is the keyword.
BY : is the keyword.
field_name : column on which you want to sort the relation.
ASC : sort in ascending order
DESC : sort in descending order.
Eg:
We will sort the relation A based on the ratings field and get top 3 employee details with highest ratings.

Sorted = ORDER A by ratings DESC;



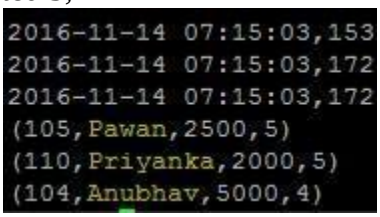
Result = LIMIT Sorted 3;

You can also Order the relation based on multiple fields. Let's order the relation A based on Descending 'ratings' and Ascending 'names' and generate top 3 result.

Ex:

Double_sorted = ORDER A by ratings DESC, name ASC;

Final_result = LIMIT Double_sorted 3;



Now, compare the 'Result' and 'Final_result' relation.

Group

Groups the data based on one or multiple fields. It groups together tuples that have the same group key (key field). The key field will be a tuple if the group key has more than onefield, otherwise it will be the same type as that of the group key.

Syntax:

alias = GROUP alias {ALL | BY field};

Where;

alias : is the relation

GROUP : is the keyword

ALL : keyword. Use ALL if you want all tuples to go to a single group

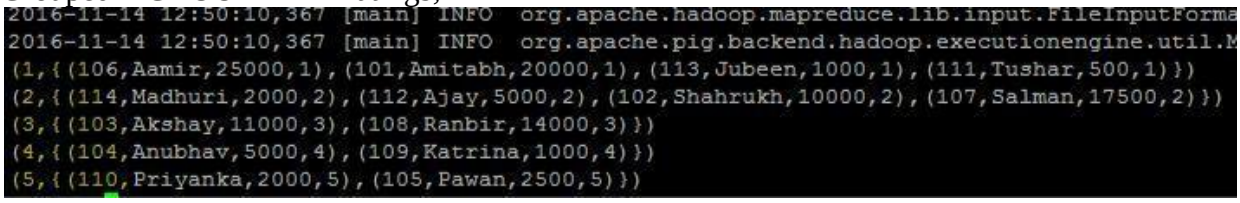
BY : keyword

Field : field name on which you want to group your data.

Ex:

We will group our relation A based on ratings.

Grouped = GROUP A BY ratings;



You can see that the output is a tuple based on 'ratings' with multiple bags. You can also group the data based on more than one fields. For example,

Multi_group = GROUP A BY (ratings, salary);

Foreach

It generates data transformations based on desired columns of data.
Syntax:
alias = FOREACH alias GENERATE {expression |
field}; Where;
alias : is the relation
FOREACH : is the keyword
GENERATE : is the keyword
Ex:
In our previous example we saw how to group a relation. Now, using FOREACH, we will generate the count of employees belonging to a particular group.
Result = FOREACH Grouped GENERATE group,COUNT(A.ratings);

```
grunt> grouped = GROUP A BY ratings;
grunt> describe grouped
grouped: {group: int,A: {(id: int,name: chararray,salary: int,ratings: int)}}
grunt> result = FOREACH grouped GENERATE group,COUNT(A.ratings);
```

```
2016-11-14 14:03:26,495 [main
2016-11-14 14:03:26,516 [main
2016-11-14 14:03:26,516 [main
(1,4)
(2,4)
(3,2)
(4,2)
(5,2)
```

From the result, we can conclude that there are 4 employees who got 1 as their rating. It is indicated by the first row. Basically, if want to operate at column level, you can use Foreach.

Filter

Filters a relation based on certain condition.
Syntax:
alias = FILTER alias BY expression;
Where;
alias : is the relation
FILTER : is the keyword
BY : is the keyword
expression : condition on which filter will be performed.
Ex:
We will filter our data (relation A) based on ratings greater than or equal to 4.
Filtered = FILTER A BY ratings >= 4;

```
2016-11-14 15:15:29,881 [main] WARN
2016-11-14 15:15:29,899 [main] INFO
2016-11-14 15:15:29,899 [main] INFO
(104,Anubhav,5000,4)
(105,Pawan,2500,5)
(109,Katrina,1000,4)
(110,Priyanka,2000,5)
```

Use the FILTER operator to work with tuples or rows of data (if you want to work with columns of data, use the FOREACH …GENERATE operation).
FILTER is commonly used to select the data that you want; or, conversely, to filter out (remove) the data you don't want.
We can also use multiple conditions to filter data at one go.
Eg:

Multi_condition = FILTER A BY (ratings >= 4) AND (salary >1000)
This will produce results that follows in the category of ratings greater than equals 4 and salary greater than 1000.

```
grunt> multi_condition = FILTER A BY (ratings >= 4) AND (salary > 1000);
grunt>
```

```
2016-11-14 15:42:41,182 [main] INFO
(104,Anubhav,5000,4)
(105,Pawan,2500,5)
(110,Priyanka,2000,5)
```

Store

Stores and saves the data into a filesystem.
Syntax:
STORE alias INTO ‘directory’ [USING function] Where;
STORE : is a keyword
alias : is the relation which you want to store.
INTO : is the keyword
directory : name of directory where you want to store your result.
NOTE: If directory already exists, you will receive an error and STORE operation will fail.
function : the store function. By default, PigStorage is the default storage function and hence it is not mandatory to mention this explicitly.
Ex:
We will store our result named “Multi_condition” (achieved in previous operation) into local file system.

```
grunt> STORE multi_condition INTO '/home/acadgild/pig/myoutput' USING PigStorage('|');
```

Only catch here is, I have specified PigStorage(‘|’) , means, I will loading result into local fs and the delimiter will be pipeline i.e. ‘|’
Let’s check the result.

```
[acadgild@localhost pig]$ pwd
/home/acadgild/pig
[acadgild@localhost pig]$ ls
employee_details.txt  employee_expenses.txt  myoutput
[acadgild@localhost pig]$ cd myoutput/
[acadgild@localhost myoutput]$ ls
part-m-00000  _SUCCESS
[acadgild@localhost myoutput]$ cat part-m-00000
104|Anubhav|5000|4
105|Pawan|2500|5
110|Priyanka|2000|5
```

PRACTICAL NO – 5

Aim: Implementing Database Operations on Hive.

THEORY:

Hive defines a simple SQL-like query language to querying and managing large datasets called Hive-QL (HQL). It's easy to use if you're familiar with SQL Language. Hive allows programmers who are familiar with the language to write the custom MapReduce framework to perform more sophisticated analysis.

Uses of Hive:

1. The Apache Hive distributed storage.
2. Hive provides tools to enable easy data extract/transform/load (ETL)
3. It provides the structure on a variety of data formats.
4. By using Hive, we can access files stored in Hadoop Distributed File System (HDFS is used to querying and managing large datasets residing in) or in other data storage systems such as Apache HBase.

Limitations of Hive:

• Hive is not designed for Online transaction processing (OLTP), it is only used for the Online Analytical Processing.
• Hive supports overwriting or apprehending data, but not updates and deletes.
• In Hive, sub queries are not supported.

Why Hive is used inspite of Pig?

The following are the reasons why Hive is used in spite of Pig's availability:

- Hive-QL is a declarative language line SQL, PigLatin is a data flow language.
- Pig: a data-flow language and environment for exploring very large datasets.
- Hive: a distributed data warehouse.

Components of Hive:

Metastore :

Hive stores the schema of the Hive tables in a Hive Metastore. Metastore is used to hold all the information about the tables and partitions that are in the warehouse. By default, the metastore is run in the same process as the Hive service and the default Metastore is DerBy Database.

SerDe :

Serializer, Deserializer gives instructions to hive on how to process a record.

Hive Commands :

Data Definition Language (DDL)

DDL statements are used to build and modify the tables and other objects in the database.

Example :

CREATE, DROP, TRUNCATE, ALTER, SHOW, DESCRIBE Statements.

Go to Hive shell by giving the command sudo hive and enter the command **'create database<data base name>'** to create the new database in the Hive.

```
hive> create database retail;
OK
Time taken: 5.275 seconds
hive>
```

To list out the databases in Hive warehouse, enter the command '**show databases'.**

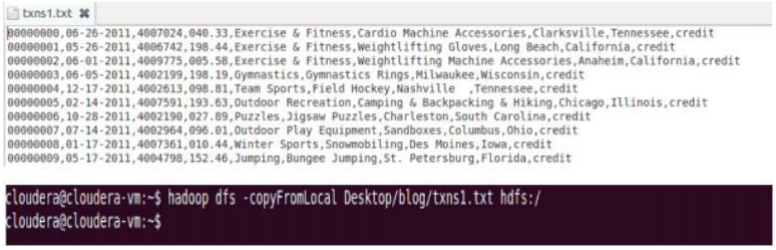
```
hive> show databases;
OK
default
retail
Time taken: 0.228 seconds
hive>
```

The database creates in a default location of the Hive warehouse. In Cloudera, Hive database store in a /user/hive/warehouse.

The command to use the database is **USE <data base name>**

```
hive> use retail;
OK
Time taken: 0.023 seconds
hive>
```

Copy the input data to HDFS from local by using the copy From Local command.



When we create a table in hive, it creates in the default location of the hive warehouse. – "/user/hive/warehouse", after creation of the table we can move the data from HDFS to hive table.

The following command creates a table with in location of "/user/hive/warehouse/retail.db"

Note : retail.db is the database created in the Hive warehouse.

```
hive> create table txnrecords(txnno INT, txndate STRING, custno INT, amount DOUBLE,category STRING, product STRING, city STRIN
G, state STRING, spendby STRING) row format delimited fields terminated by ',' stored as textfile;
OK
Time taken: 1.163 seconds
hive>
```

```
hive> describe txnrecords;
OK
txnno   int
txndate string
custno  int
amount  double
category        string
product string
city    string
state   string
spendby string
Time taken: 0.122 seconds
hive>
```

Data Manipulation Language (DML)

DML statements are used to retrieve, store, modify, delete, insert and update data in the database.

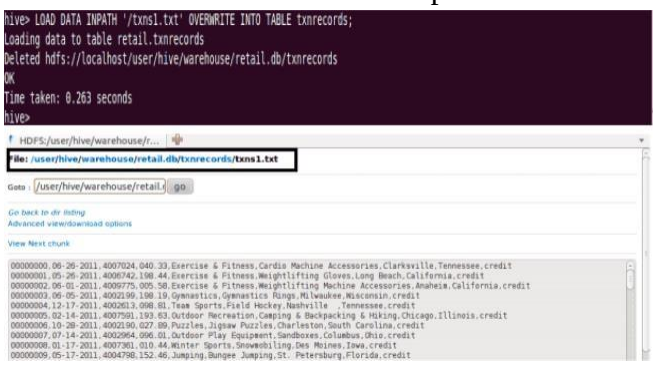
Example :

LOAD, INSERT Statements.

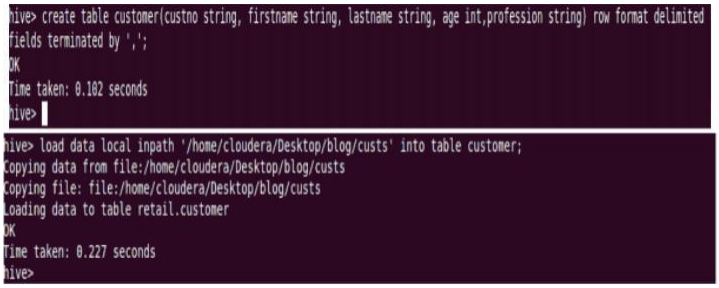
Syntax :

LOAD data <LOCAL> inpath <file path> into table [tablename]

The Load operation is used to move the data into corresponding Hive table. If the keyword **local** is specified, then in the load command will give the local file system path. If the keyword local is not specified we have to use the HDFS path of the file.



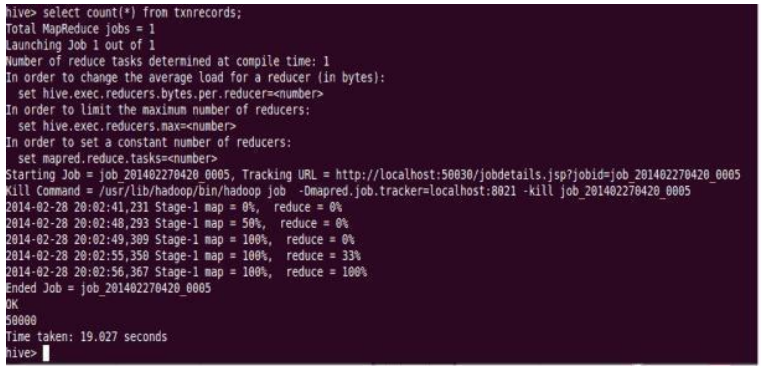
Here are some examples for the LOAD data LOCAL command



After loading the data into the Hive table we can apply the Data Manipulation Statements or aggregate functions retrieve the data.

Example to count number of records:

Count aggregate function is used count the total number of the records in a table.



'create external' Table :

The **create external** keyword is used to create a table and provides a location where the table will create, so that Hive does not use a default location for this table. An **EXTERNAL** table points to any HDFS location for its storage, rather than default storage.

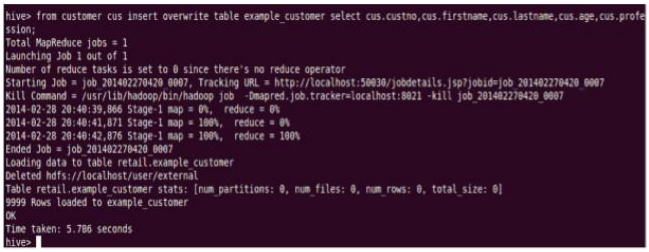
```
hive> create external table example_customer(custno string, firstname string, lastname string, age int,profession string) row
format delimited fields terminated by ',' LOCATION '/user/external';
OK
Time taken: 0.059 seconds
hive>
```

Insert Command:

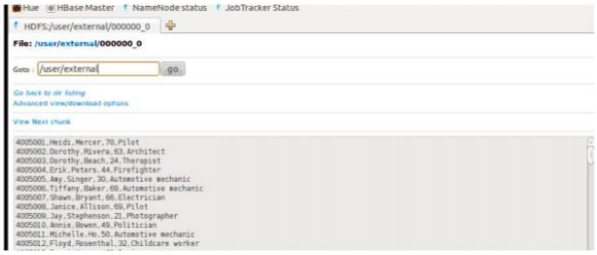
The **insert** command is used to load the data Hive table. Inserts can be done to a table or a partition.

• INSERT OVERWRITE is used to overwrite the existing data in the table or partition.

• INSERT INTO is used to append the data into existing data in a table. (Note: INSERT INTO syntax is work from the version 0.8)



```
hive> from customer cus insert overwrite table example_customer select cus.custno,cus.firstname,cus.lastname,cus.age,cus.profe
ssion;
Total MapReduce jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_201402270420_0007, Tracking URL = http://localhost:50030/jobdetails.jsp?jobid=job_201402270420_0007
Kill Command = /usr/lib/hadoop/bin/hadoop job  -Dmapred.job.tracker=localhost:8021 -kill job_201402270420_0007
2014-02-28 20:40:39,866 Stage-1 map = 0%,  reduce = 0%
2014-02-28 20:40:41,871 Stage-1 map = 100%,  reduce = 0%
2014-02-28 20:40:42,876 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_201402270420_0007
Loading data to table retail.example_customer
Deleted hdfs://localhost/user/external
Table retail.example_customer stats: [num_partitions: 0, num_files: 0, num_rows: 0, total_size: 0]
9999 Rows loaded to example_customer
OK
Time taken: 5.786 seconds
hive>
```



Example for 'Partitioned By' and 'Clustered By' Command :

'Partitioned by' is used to divided the table into the Partition and can be divided in to buckets by using the '**Clustered By**' command.

```
hive> create table txnrecsByCat(txnno INT, txndate STRING, custno INT, amount DOUBLE,product STRING, city STRING, state STRING
, spendby STRING) partitioned by (category STRING) clustered by (state) INTO 10 buckets row format delimited fields terminated
 by ',' stored as textfile;
OK
Time taken: 0.101 seconds
hive>
```

```
hive> from txnrecords txn INSERT OVERWRITE TABLE record PARTITION(category)select txn.txnno,txn.txndate,txn.custno,txn.amount,
txn.product,txn.city,txn.state,txn.spendby, txn.category;
FAILED: Error in semantic analysis: Dynamic partition strict mode requires at least one static partition column. To turn this
off set hive.exec.dynamic.partition.mode=nonstrict
```

When we insert the data Hive throwing errors, the dynamic partition mode is strict and dynamic partition not enabled (by Jeff at dresshead website). So we need to set the following parameters in Hive shell.

PRACTICAL NO – 6

Aim: Implementing Bloom Filter using Map-Reduce.

THEORY:

Bloom filtering is similar to generic filtering in that it is looking at each record and deciding whether to keep or remove it. However, there are two major differences that set it apart from generic filtering. First, we want to filter the record based on some sort of set membership operation against the hot values. For example: keep or throw away this record if the value in the user field is a member of a predetermined list of users.Second, the set membership is going to be evaluated with a Bloom filter.

Formula for optimal size of bloom filter

1. OptimalBloomFilterSize = (-The number of members in the set * **log**(The desired false positive rate))/**log**(2)^2

Formula to get the optimalK

1. OptimalK = (OptimalBloomFilterSize * **log**(2))/The number of members in the set

CODE:

We are reading the input file and storing the bloom filter hot words file in the local file system (I am using windows) ideally the file should be read and stored in the hdfs using hadoop hdfs api for simplicity purpose have not included the code for hdfs filesystem.This Bloom filter file can later be deserialized from HDFS or local system just as easily as it was written.Just open up the file using the FileSystem object and pass it to BloomFilter.readFields.

```
1. import java.io.DataOutputStream;
2. import java.io.FileOutputStream;
3. import java.io.IOException;
4. import org.apache.hadoop.util.bloom.BloomFilter;
5. import org.apache.hadoop.util.bloom.Key;
6. import org.apache.hadoop.util.hash.Hash;
7.
8. public class DepartmentBloomFilterTrainer {
9. public static int getBloomFilterOptimalSize(int numElements, float falsePosRate) {
10. return (int) (-numElements * (float) Math.log(falsePosRate) /
    Math.pow(Math.log(2), 2));
11. }
12. public static int getOptimalK(float numElements, float vectorSize) {
13. return (int) Math.round(vectorSize * Math.log(2) / numElements);
14. }
15. public static void main(String[] args) throws IOException {
16. args = new String[] { "32658", "0.2","Replace this string with Input file location",
17. "Replace this string with output path location where the bloom filter hot list data will
    be stored","" };
18. int numMembers = Integer.parseInt(args[0]);
```

```
19. float falsePosRate = Float.parseFloat(args[1]);
20. int vectorSize = getBloomFilterOptimalSize(numMembers, falsePosRate);
21. int nbHash = getOptimalK(numMembers, vectorSize);
22. BloomFilter filter = new BloomFilter(vectorSize, nbHash, Hash.MURMUR_HASH);
23. ConfigFile configFile = new ConfigFile(args[2], FileType.script, FilePath.absolutePath);
24. String fileContent = configFile.getFileContent();
25. String[] fileLine = fileContent.split("\n");
26. for (String lineData : fileLine) {
27. String lineDataSplit[] = lineData.split(",", -1);
28. String departmentName = lineDataSplit[3];
29. filter.add(new Key(departmentName.getBytes()));
30. }
31. DataOutputStream dataOut = new
    DataOutputStream(new FileOutputStream(args[3]));
32. filter.write(dataOut);
33. dataOut.flush();
34. dataOut.close();
35. }
36. }
```

In the setup method the bloom fileter file is deserialized and loaded into the bloom filter.In the map method, the departmentName is extracted from each input record and tested against the Bloom filter. If the word is a member, the entire record is output to the file system.Ideally to load the bloom filter hot words we should be using DistributedCache a hadoop utility that ensures that a file in HDFS is present on the local file system of each task that requires that file for simplicity purpose i am loading it from my local file system. As we have trained the bloom filter with PUBLIC LIBRARY department the output of the map reduce program will have only employee data relevant to PUBLIC LIBRARY department.

```
1. import java.io.DataInputStream;
2. import java.io.FileInputStream;
3. import java.io.IOException;
4. import org.apache.hadoop.io.NullWritable;
5. import org.apache.hadoop.io.Text;
6. import org.apache.hadoop.mapreduce.Mapper;
7. import org.apache.hadoop.util.bloom.BloomFilter;
8. import org.apache.hadoop.util.bloom.Key;
9.
10. public class DepartmentBloomFilterMapper extends Mapper<Object, Text, Text,
    NullWritable> {
11. private BloomFilter filter = new BloomFilter();
12. protected void setup(Context context) throws IOException, InterruptedException {
13. DataInputStream dataInputStream = new DataInputStream(
```

```
14. new FileInputStream(context.getConfiguration().get("bloom_filter_file_location")));
15. filter.readFields(dataInputStream);
16. dataInputStream.close();
17. }
18. public void map(Object key, Text value, Context context) throws
    IOException, InterruptedException {
19. String data = value.toString();
20. String[] field = data.split(",", -1);
21. String department = null;
22. if (null != field && field.length == 9 && field[3].length() > 0) {
23. department = field[3];
24. if (filter.membershipTest(new Key(department.getBytes()))) {
25. context.write(value, NullWritable.get());
26. }
27. }
28. }
29. }
```

Bloom filters can assist expensive operations by eliminating unnecessary ones. For example a Bloom filter can be previously trained with IDs of all users that have a salary of more than x and use the Bloom filter to do an initial test before querying the database to retrieve more information about each employee. By eliminating unnecessary queries, we can speed up processing time.

Finally we will use the driver class to test everything is working fine as expected . The output will contain only the data of employees who belongs to the department PUBLIC LIBRARY which can be used for further analysis.

```
1. import java.io.File;
2. import org.apache.commons.io.FileUtils;
3. import org.apache.hadoop.conf.Configuration;
4. import org.apache.hadoop.fs.Path;
5. import org.apache.hadoop.io.NullWritable;
6. import org.apache.hadoop.io.Text;
7. import org.apache.hadoop.mapreduce.Job;
8. import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
10. import com.hadoop.design.summarization.blog.ConfigurationFactory;
11.11.
12. public class BloomFilterDriver {
13. public static void main(String[] args) throws Exception {
14. args = new String[] { "Replace this string with Input Path location",
15. "Replace this string with output Path location","Replace this string with Input file
    location of bloom filter hot list" };
```

```
16. FileUtils.deleteDirectory(new File(args[1]));
17. System.setProperty("hadoop.home.dir", "Replace this string with hadoop home directory
    location");
18. if (args.length != 3) {
19. System.err.println("Please specify the input and output path");
20. System.exit(-1);
21. }
22. Configuration conf = ConfigurationFactory.getInstance();
23. conf.set("bloom_filter_file_location",args[2]);
24. Job job = Job.getInstance(conf);
25. job.setJarByClass(BloomFilterDriver.class);
26. job.setJobName("Bloom_Filter_Department");
27. FileInputFormat.addInputPath(job, new Path(args[0]));
28. FileOutputFormat.setOutputPath(job, new Path(args[1]));
29. job.setMapperClass(DepartmentBloomFilterMapper.class);
30. job.setOutputKeyClass(Text.class);
31. job.setOutputValueClass(NullWritable.class);
32. System.exit(job.waitForCompletion(true) ? 0 : 1);
33.
34. }
35. }
```

PRACTICAL NO – 7

Aim: Implementing Frequent Item Set Algorithm Using Map-Reduce.

THEORY:

Frequent Itemset Mining aims to find the regularities in transaction dataset. Map Reduce maps the presence of set of data items in a transaction and reduces the Frequent Item set with low frequency. The input consists of a set of transactions and each transaction contains several items. The Map function reads the items from each transaction and generates the output with key and value. Key is represented with item and value is represented by 1. After map phase is completed, reduce function is executed and it aggregates the values corresponding to key. From the results, the frequent items are computed on the basis of minimum support value.

CODE-

Mapper Function

Map Phase input:<k1, v1>
k1-Line no
v1-Transaction
// get items from each
transaction //itemcount set to 1
for each
item k2-item
v2-1
End for
Output(k2, v2)

Reducer Function

//Count the occurrences for each item
// minimum support
Reduce Phase input:<k2, List<v2>>
sum the value for each item occurrence
if(occurrence of an item satisfy minsup)
Emit(Frequent item)
k3-Frequent item
v3-occurrences
Output: <k3,v3>

```
import java.io.BufferedReader;
import java.io.*;
import java.io.IOException;
import java.net.*;
import java.util.ArrayList;
import java.util.*;

import model.HashTreeNode;
import model.ItemSet;
```

```
import model.Transaction;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import utils.AprioriUtils;
import utils.HashTreeUtils;
import org.apache.hadoop.fs.*;
/*
 * A parallel hadoop-based Apriori
 algorithm */
public class MRApriori extends Configured implements
Tool {
	private static String jobPrefix = "MRApriori Algorithm Phase ";
	// TODO : This is bad as I using a global shared variable between
functions which should
	// ideally be a function parameter. Need to fix this later. These parameters
are required in
	// reducer logic and have to be dynamica. How can I pass some
initialisation parameters to
	// reducer ?

	public int run(String[] args) throws IOException,
InterruptedException, ClassNotFoundException {
		if(args.length != 5) {
			System.err.println("Incorrect number of command line args.
Exiting !!");
			return -1;
		}


		String hdfsInputDir = args[0];
		String hdfsOutputDirPrefix = args[1];

		int maxPasses = Integer.parseInt(args[2]);
		Double MIN_SUPPORT_PERCENT = Double.parseDouble(args[3]);
```

```
		Integer MAX_NUM_TXNS = Integer.parseInt(args[4]);

		System.out.println("InputDir		: " + hdfsInputDir);
		System.out.println("OutputDir Prefix : " + hdfsOutputDirPrefix);
		System.out.println("Number of Passes : " + maxPasses);
		System.out.println("MinSupPercent : " + MIN_SUPPORT_PERCENT);
		System.out.println("Max Txns : " + MAX_NUM_TXNS);
		long startTime = System.currentTimeMillis();
		long endTime = System.currentTimeMillis();
		for(int passNum=1; passNum <= maxPasses; passNum++) {
			endTime = System.currentTimeMillis();
			boolean isPassKMRJobDone = runPassKMRJob(hdfsInputDir,
hdfsOutputDirPrefix, passNum, MIN_SUPPORT_PERCENT, MAX_NUM_TXNS);
			if(!isPassKMRJobDone) {
				System.err.println("Phase1 MapReduce job failed.
Exiting !!");
				return -1;
			}
			System.out.println("For pass " + passNum + " = " +
(System.currentTimeMillis() - endTime));
		}
		endTime = System.currentTimeMillis(); System.out.println("Total
		time taken = " + (endTime - startTime));

		return 1;
	}
	private static boolean runPassKMRJob(String hdfsInputDir, String
hdfsOutputDirPrefix, int passNum, Double MIN_SUPPORT_PERCENT,
Integer MAX_NUM_TXNS)
			throws IOException, InterruptedException,
ClassNotFoundException
	{
		boolean isMRJobSuccess = false;

		Configuration passKMRConf = new Configuration();
		passKMRConf.setInt("passNum", passNum);
		passKMRConf.set("minSup",
Double.toString(MIN_SUPPORT_PERCENT));
		passKMRConf.setInt("numTxns", MAX_NUM_TXNS);
		System.out.println("Starting AprioriPhase" + passNum +"Job");
		if(passNum > 1) {
			DistributedCache.addCacheFile( URI.create("hdfs://127.0.0.1:54310"
					+
hdfsOutputDirPrefix +
					(passNum-1) + "/part-r-00000"), passKMRConf
			);
```

```
                        System.out.println("Added to distributed cache the output of
pass " + (passNum-1));
                }
                */
                Job aprioriPassKMRJob = new Job(passKMRConf, jobPrefix +
passNum);
                if(passNum == 1) {
                        configureAprioriJob(aprioriPassKMRJob,
AprioriPass1Mapper.class);
                }
                else {
                        configureAprioriJob(aprioriPassKMRJob,
AprioriPassKMapper.class);
                }
                FileInputFormat.addInputPath(aprioriPassKMRJob, new
Path(hdfsInputDir));
                System.out.println("saurabh " + new Path(hdfsInputDir));
                FileOutputFormat.setOutputPath(aprioriPassKMRJob, new
Path(hdfsOutputDirPrefix + passNum));

                isMRJobSuccess = (aprioriPassKMRJob.waitForCompletion(true) ? true
: false);
                System.out.println("Finished AprioriPhase" + passNum +"Job");

                return isMRJobSuccess;
        }


        @SuppressWarnings({ "unchecked", "rawtypes" })
        private static void configureAprioriJob(Job aprioriJob, Class mapperClass)
        {
                aprioriJob.setJarByClass(MRApriori.class);
                aprioriJob.setMapperClass(mapperClass);
                aprioriJob.setReducerClass(AprioriReducer.class);
                aprioriJob.setOutputKeyClass(Text.class);
                aprioriJob.setOutputValueClass(IntWritable.class);
        }
        //---------------------Utility functions ----------------------------------------
        // Phase1 - MapReduce
        public static class AprioriPass1Mapper extends Mapper<Object, Text,
Text, IntWritable> {
                private final static IntWritable one = new IntWritable(1);
                private Text item = new Text();
                public void map(Object key, Text txnRecord, Context context)
throws IOException, InterruptedException {
                        Transaction txn =
AprioriUtils.getTransaction(txnRecord.toString());
```

```
                for(Integer itemId:txn.getItems()){
                        item.set(itemId.toString());
                        context.write(item, one);
                    }
                }
        }
        public static class AprioriReducer extends Reducer<Text, IntWritable,
Text, IntWritable> {
                public void reduce(Text itemset, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
                    int countItemId = 0;
                    for (IntWritable value : values) {
                        countItemId += value.get();
                    }
                    // TODO : This can be improved. Creating too many
                    strings. String itemsetIds = itemset.toString();
                    itemsetIds  =  itemsetIds.replace("[",  "");
                    itemsetIds  =  itemsetIds.replace("]",  "");
                    itemsetIds  =  itemsetIds.replace("  ",  "");
                    Double minSup =
Double.parseDouble(context.getConfiguration().get("minSup"));
                    Integer numTxns =
context.getConfiguration().getInt("numTxns", 2);
                    //System.out.println("dsfsdfsdf: " + MIN_SUPPORT_PERCENT
+ " " + MAX_NUM_TXNS);
                    // If the item has minSupport, then it is a large itemset.
                    if(AprioriUtils.hasMinSupport(minSup, numTxns, countItemId))
{
                        context.write(new Text(itemsetIds), new
IntWritable(countItemId));
                    }

                }
        }

        // Phase2 - MapReduce
        public static class AprioriPassKMapper extends Mapper<Object, Text,
Text, IntWritable> {
                private final static IntWritable one = new IntWritable(1);
                private Text item = new Text();
                private List<ItemSet> largeItemsetsPrevPass = new
ArrayList<ItemSet>();
                private List<ItemSet> candidateItemsets     = null;
                private HashTreeNode hashTreeRootNode       = null;


                @Override
                public void setup(Context context) throws IOException {
```

```
                        //Path[] uris =
DistributedCache.getLocalCacheFiles(context.getConfiguration());
                        int passNum = context.getConfiguration().getInt("passNum",
                        2); String opFileLastPass =
context.getConfiguration().get("fs.default.name") + "/user/hduser/mrapriori-out-" +
(passNum-1) + "/part-r-00000";
                        //System.out.println("ahsdkjdsgfjhgf" + opFileLastPass);
                        //System.out.println("Distributed cache file to search " +
opFileLastPass);

                        try
                        {
                                Path pt=new Path(opFileLastPass);
                        FileSystem fs = FileSystem.get(context.getConfiguration());
                        BufferedReader fis=new BufferedReader(new
InputStreamReader(fs.open(pt)));
                                String currLine = null;

        //System.out.println("aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaa");
                                while ((currLine = fis.readLine()) != null) {
                                        currLine = currLine.trim();
                                        String[] words = currLine.split("[\\s\\t]+");
                                        if(words.length < 2) {
                                                continue;
                                        }

                                        List<Integer> items = new ArrayList<Integer>();
                                        for(int k=0; k < words.length -1 ; k++){
                                                String csvItemIds = words[k];
                                                String[] itemIds = csvItemIds.split(",");
                                                for(String itemId : itemIds) {

        items.add(Integer.parseInt(itemId));
                                                }
                                        }
                                        String finalWord = words[words.length-1];
                                        int supportCount = Integer.parseInt(finalWord);
                                        //System.out.println(items + " --> " +
supportCount);
                                        largeItemsetsPrevPass.add(new ItemSet(items,
supportCount));
                                }
                        }
                        catch(Exception e)
                        {
```

```
                        }

                        candidateItemsets =
        AprioriUtils.getCandidateItemsets(largeItemsetsPrevPass, (passNum-1));
                        hashTreeRootNode =
        HashTreeUtils.buildHashTree(candidateItemsets, passNum); // This would be changed
        later
                }
                public void map(Object key, Text txnRecord, Context context) throws
        IOException, InterruptedException {
                        Transaction txn =
        AprioriUtils.getTransaction(txnRecord.toString()); List<ItemSet>
                        candidateItemsetsInTxn =
        HashTreeUtils.findItemsets(hashTreeRootNode, txn, 0);
                        for(ItemSet itemset : candidateItemsetsInTxn) {
                                item.set(itemset.getItems().toString());
                                context.write(item, one);
                        }
                }
        }
        public static void main(String[] args) throws Exception
        {
                int exitCode = ToolRunner.run(new MRApriori(),
                args); System.exit(exitCode);
        }
}
```

OUTPUT-

- OUTPUT1

  2 3

  3 3

  5 3

- OUTPUT2

  1,3 2

  2,3 2

  2,5 3

  3,5 2

  2,3,5 2

PRACTICAL NO – 8

Aim: Implementing Clustering Algorithm Using Map-Reduce

Theory:

MapReduce runs as a series of jobs, with each job essentially a separate Java application that goes out into the data and starts pulling out information as needed. Based on the MapReduce design, records are processed in isolation via tasks called Mappers. The output from the Mapper tasks is further processed by a second set of tasks, the Reducers, where the results from the different Mapper tasks are merged together. The Map and Reduce functions of MapReduce are both defined with respect to data structured in (key, value) pairs. Map takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain:

Map(k1,v1) → list(k2,v2)

The Map function is applied in parallel to every pair in the input dataset. This produces a list of pairs for each call. After that, the MapReduce framework collects all pairs with the same key from all lists and groups them together, creating one group for each key. The Reduce function is then applied in parallel to each group, which in turn produces a collection of values in the same domain:

Reduce(k2, list (v2)) → list(v3)

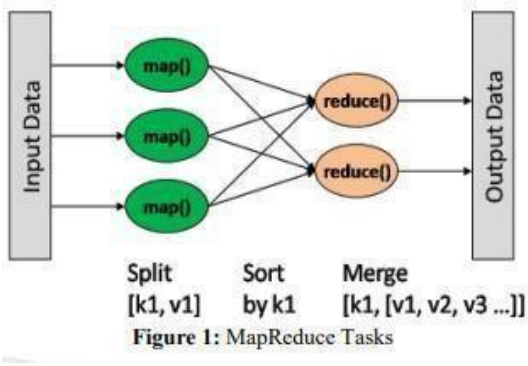


Figure 1: MapReduce Tasks

Algorithm for Mapper

Input: A set of objects X = {x1, x2… xn}, A Set ofinitial Centroids C = {c1, c2, ,ck}

Output: An output list which contains pairs of (Ci, xj)where $1 \leq i \leq n$ and $1 \leq j \leq k$

Procedure

M1←{x1, x2… xm}

current_centroids←C

Distance (p, q) $=\sqrt{\Sigma d_{i=1}}(pi - qi)^2$ (where pi (or qi)is the coordinate of p (or q) in dimension

i) for all xi ϵ M1 such that $1 \leq i \leq m$ do

    bestCentroid←null

    minDist←∞

    for all c ϵ current_centroids do

```
                dist← distance (xi, c)
                if (bestCentroid = null || dist<minDist)
                then
                        minDist←dist
                                bestCentroid ← c
                        end if
        end for
        emit (bestCentroid xi)
        i+=1
    end for
    return Outputlist
```

Algorithm for Reducer
Input: (Key, Value), where key = bestCentroid and Value =Objects assigned to the lpgr'; 1\] x centroid by the mapper
Output: (Key, Value), where key = oldCentroid and value = newBestCentroid which is the new centroid value calculated for that bestCentroid
Procedure

```
Outputlist←outputlist from mappers
← { }
newCentroidList ← null
for all β outputlist do
        centroid ←β.key
        object ←β.value
        [centroid] ← object
end for
for all centroid ϵ    do
        newCentroid, sumofObjects,
        sumofObjects← null
        for all object ϵ    [centroid] do
                sumofObjects += object
                numofObjects += 1
        end for
newCentroid ← (sumofObjects +
numofObjects)
emit (centroid, newCentroid)
end for
end
```

The outcome of the k-means map reduce algorithm is the cluster points along with bounded documents as <key, value> pairs, where key is the cluster id and value contains in the form of vector: weight. The weight indicates the probability of vector be a point in that cluster. For Example: Key: 92: Value: 1.0: [32:0.127,79:0.114, 97:0.114, 157:0.148 ...].

The final output of the program will be the cluster name, filename: number of text documents that belong to that cluster

PRACTICAL NO – 9

Aim: Implementing Page Rank Algorithm Using Map-Reduce.

Theory:

PageRank is a way of measuring the importance of website pages. PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

In the general case, the PageRank value for any page u can be expressed as:

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

i.e. the PageRank value for a page u is dependent on the PageRank values for each page v contained in the set Bu (the set containing all pages linking to page u), divided by the number L(v) of links from page v.

Suppose consider a small network of four web pages: A, B, C and D. Links from a page to itself, or multiple outbound links from one single page to another single page, are ignored. PageRank is initialized to the same value for all pages. In the original form of PageRank, the sum of PageRank over all pages was the total number of pages on the web at that time, so each page in this example would have an initial value of 1.

$$PR(A) = \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)}.$$

The damping factor (generally set to 0.85) is subtracted from 1 (and in some variations of the algorithm, the result is divided by the number of documents (N) in the collection) and this term is then added to the product of the damping factor and the sum of the incoming PageRank scores. That is,

$$PR(A) = \frac{1-d}{N} + d\left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots\right).$$

So any page's PageRank is derived in large part from the PageRanks of other pages.
The damping factor adjusts the derived value downward.

CODE:

```
import numpy as np
import scipy as sc
import pandas as pd
from fractions import Fraction
def display_format(my_vector, my_decimal):
  return np.round((my_vector).astype(np.float), decimals=my_decimal)
my_dp = Fraction(1,3)
Mat = np.matrix([[0,0,1],
      [Fraction(1,2),0,0],
      [Fraction(1,2),1,0]])
Ex = np.zeros((3,3))
```

```
Ex[:] = my_dp
beta = 0.7
Al = beta * Mat + ((1-beta) * Ex)
r = np.matrix([my_dp, my_dp, my_dp])
r = np.transpose(r)
previous_r = r
for i in range(1,100):
   r = Al * r
   print (display_format(r,3))
   if (previous_r==r).all():
      break
   previous_r = r
print ("Final:\n", display_format(r,3))
print ("sum", np.sum(r))
```

OUTPUT:

```
[[0.333]
 [0.217]
 [0.45 ]]
[[0.415]
 [0.217]
 [0.368]]
[[0.358]
 [0.245]
 [0.397]]
...
//Reduce upper matrix if need to
...
[[0.375]
 [0.231]
 [0.393]]
```

FINAL:

```
 [[0.375]
 [0.231]
 [0.393]]
sum 0.9999999999999951
```

PRACTICAL NO – 10

MINI PROJECT:

Description:

Few topics for Projects:

a. Twitter data analysis

b. Fraud Detection

c. Text Mining

d. Equity Analysis etc.

Few websites for sample data:

www.data.gov.in

www.nseindia.in

www.censusindia.gov.in

www.importexportdata.in