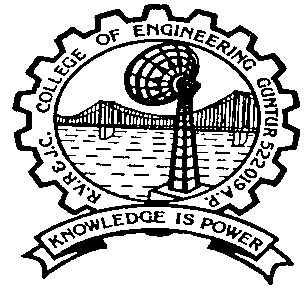
**DEPARTMENT OF COMPUTER APPLICATIONS**

**MCA 507: Data Mining and Hadoop Lab**

**III/III MCA - (V SEMESTER)**

**Lab Manual**

****

**R.V.R. & J.C.COLLEGE OF ENGINEERING**

(Approved by A.I.C.T.E.)

(Affiliated to Acharya Nagarjuna University)

Chandramoulipuram :: Chowdavaram

GUNTUR – 522019 :: ANDHRA PRADESH

1. **PROFORMA FOR THEORY BASED COURSE DESCRIPTION**

Course Code : MCA-507

Course Title : Data Mining and Hadoop Lab

Year & Semester : III/III MCA (V Semester)

Periods/Week : 06

Nature of the Course : Core

Name of the Instructor-I : I.Naga Padmaja

Designation : Assistant Professor

Mail : nagapadmaja.indeti@gmail.com

Name of the Instructor-II : M.Brahmaiah

Designation : Assistant Professor

Mail : madamanchib@gmail.com

1. **COURSE CONTENT**

**MCA 507: Data Mining and Hadoop Lab**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Lectures | : | 6 periods/week | Sessional Marks | : | 30 |
| University Examination Duration | : | 3 hours | University Examination Marks | : | 70 |

**Lab Cycle**

**Data Mining (Using Java, WEKA or any open source data mining tool)**

1. Write a program to Generate Association rules by using Apriori Algorithm
2. Write a program to implement naïve Bayesian classification
3. Write a program to implement k-means clustering algorithm
4. Write a program to implement k-mediods clustering algorithm
5. Write a program to implement dbscan algorithm

**Hadoop**

1. Implement the following Data structures in Java

a)Linked Lists b) Stacks c) Queues d) Set e) Map

1. Study and configure hadoop for big data
2. Hadoop commands
3. Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.
4. Implement Matrix Multiplication with Hadoop Map Reduce
5. **Write a program to Generate Association rules by using Apriori Algorithm**

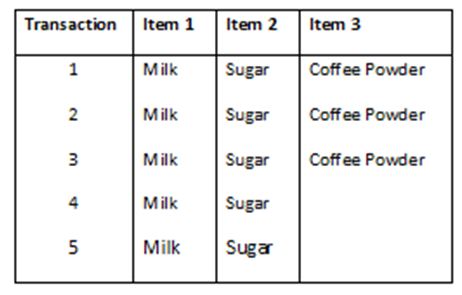
## The Approach(Apriori Algorithm)

When you go to a store, would you not want the aisles to be ordered in such a manner that reduces your efforts to buy things?

For example, I would want the toothbrush, the paste, the mouthwash  & other dental products on a single aisle – because when I buy, I tend to buy them together. This is done by a way in which we find associations between items.

In order to understand the concept better, let’s take a simple dataset (let’s name it as Coffee dataset) consisting of a few hypothetical transactions. We will try to understand this in simple English.

The Coffee dataset consisting of items purchased from a retail store.

Coffee dataset:

**The Association Rules:**

For this dataset, we can write the following association rules: (Rules are just for illustrations and understanding of the concept. They might not represent the actuals).

**Rule 1:** If Milk is purchased, then Sugar is also purchased.

**Rule 2:**  If Sugar is purchased, then Milk is also purchased.

**Rule 3:** If Milk and Sugar are purchased, Then Coffee powder is also purchased in 60% of the transactions.

Generally, association rules are written in “IF-THEN” format. We can also use the term “Antecedent” for IF (LHS) and “Consequent” for THEN (RHS).

From the above rules, we understand the following explicitly:

1. Whenever Milk is purchased, Sugar is also purchased or vice versa.
2. If Milk and Sugar are purchased then the coffee powder is also purchased. This is true in 3 out of the 5 transactions.

### Handling and Readying The Dataset

The first part of any analysis is to bring in the dataset. We will be using an inbuilt dataset “Groceries” from the ‘arules’ package to simplify our analysis.

All stores and retailers store their information of transactions in a specific type of dataset called the “Transaction” type dataset.

*The ‘pacman’ package is an assistor to help load and install the packages. we will be using pacman to load the arules package.*

The p\_load() function from “pacman” takes names of packages as arguments.

If your system has those packages, it will load them and if not, it will install and load them.

### Structural Overview and Prerequisites

Before we begin applying the “Apriori” algorithm on our dataset, we need to make sure that it is of the type “Transactions”.

The structure of our transaction type dataset shows us that it is internally divided into three slots: Data, itemInfo and itemsetInfo.

The slot “Data” contains the dimensions, dimension names and other numerical values of number of products sold by every transaction made.

## Implementing Apriori Algorithm and Key Terms and Usage

rules <- apriori(Groceries,

parameter = list(supp = 0.001, conf = 0.80))

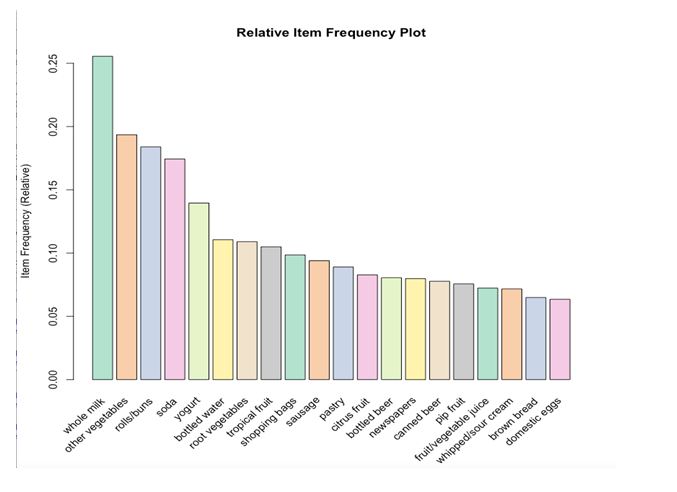
## Interpretations and Analysis

Let us first identify which products were sold how frequently in our dataset.

 The Item Frequency Histogram

These histograms depict how many times an item has occurred in our dataset as compared to the others.

The relative frequency plot accounts for the fact that “Whole Milk” and “Other Vegetables” constitute around half of the transaction dataset; half the sales of the store are of these items.



### Graphical Representation

Moving forward in the visualisation, we can use a graph to highlight the support and lifts of various items in our repository but mostly to see which product is associated with which one in the sales environment.

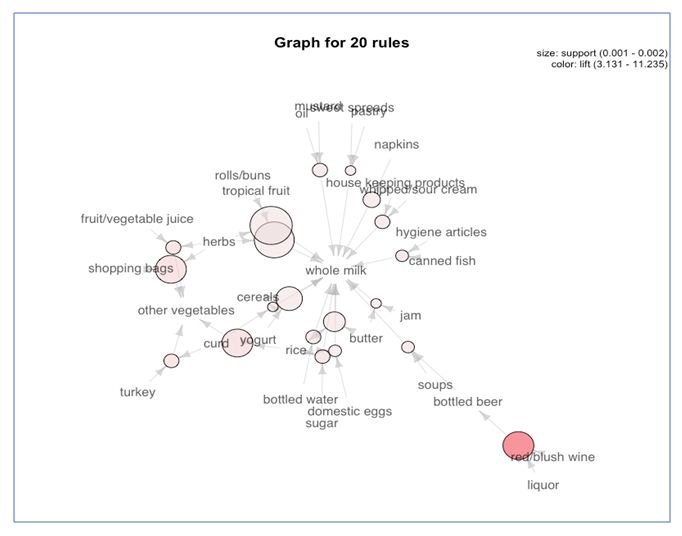
plot(rules[1:20],

method = "graph",

control = list(type = "items"))

This representation gives us a graph model of items in our dataset.

The size of graph nodes is based on support levels and the colour on lift ratios. The incoming lines show the Antecedants or the LHS and the RHS is represented by names of items.



### Individual Rule Representation

The next plot offers us a parallel coordinate system of visualisation. It would help us clearly see that which products along with which ones, result in what kinds of sales.

### Interactive Scatterplot

These plots show us each and every rule visualised into a form of a scatterplot. The confidence levels are plotted on the Y axis and Support levels on the X axis for each rule. We can hover over them in our interactive plot to see the rule.

1. **Aim: Write a program to implement Naïve Bayesian classification**

**Step 1:** Download and Install Weka

          Link: <http://www.cs.waikato.ac.nz/ml/weka/downloading.html>

**Step 2:** Download sample data

          Link: [http://www.cs.cornell.edu/people/pabo/movie-review- data/mix20\_rand700\_tokens\_cleaned.zip](http://www.cs.cornell.edu/people/pabo/movie-review-%20%20%20%20%20data/mix20_rand700_tokens_cleaned.zip)

**Step 3:** Convert data into arff format

i) Run Weka

          ii) Choose “Simple CLI”

iii) Run the command in the command line:

java weka.core.converters.TextDirectoryLoader -dir "P:\tokens" > "P:\weka

data\sample.arff"

**Step 4:** Pre-process data and save processed data

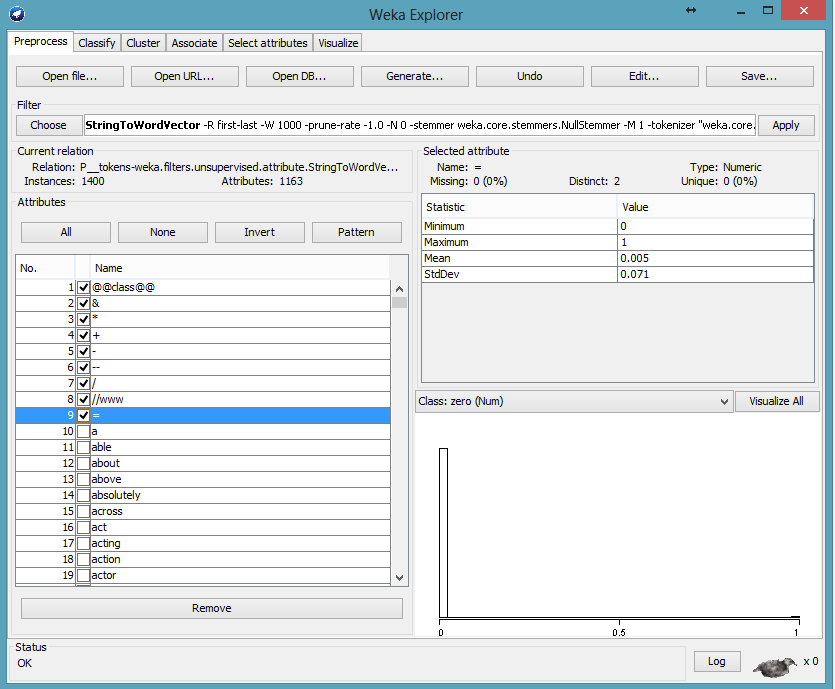
i) Choose “Explorer” from "Weka GUI Chooser"

ii) Load the arff file using “open file” option

iii) Choose a filter “StringToWordVector” and then "apply"

iv) Remove unexpected “words” using “pattern” and then “remove”

v) Remove other unexpected words manually

[](http://1.bp.blogspot.com/-smFWVXrNlak/VRJ_oUf9wJI/AAAAAAAAAn4/QbMy3rhv1iI/s1600/unexpected+words.png)

vi) Save the processed data using “save” option

**Step 5:** Run Classifier (as example : Naive Bayes)

          i) Select Classifier and method

   ii) Select evaluation option (click right to choose option)

**Step 6:** Result

i) Set output options using "more option" and run the classifier using "start"

ii) Save Result

1. **Write a program to implement k-means clustering algorithm**

public void clusterData(){

kmeans = new SimpleKMeans();

kmeans.setSeed(10);

try {

kmeans.setPreserveInstancesOrder(true);

kmeans.setNumClusters(10);

kmeans.buildClusterer(cpu);

int[] assignments = kmeans.getAssignments();

int i = 0;

for(int clusterNum : assignments) {

System.out.printf("Instance %d -> Cluster %d\n", i, clusterNum);

i++;

}

} catch (Exception e1) {

}

}

1. **Write a program to implement k-mediods clustering algorithm**

## Input:

k: the number of clusters,

D: a data set containing n objects.

Output: A set of k clusters.

## Method:

(1) Arbitrarily choose k objects in D as the initial representative objects or seeds;

(2) Repeat

(3) Assign each remaining object to the cluster with the nearest representative object;

(4) Randomly select a no representative object, o random;

(5) Compute the total cost, S, of swapping representative object, oj, with o random;

(6) If S < 0 then swap oj with o random to form the new set of k representative objects;

(7) Until no change;

Where E is the sum of the absolute error for all objects in the data set; p is the point in space representing a given object in cluster Cj; and oj is the representative object of Cj. In general, the algorithm iterates until, eventually, each representative object is actually the medoids, or most centrally located object, of its cluster. This is the basis of the k-medoids method for grouping n objects into k clusters.[6]

1. **Write a program to implement dbscan algorithm**

public int clusterInstance(Instance instance) throws Exception {

if (processed\_InstanceID >= database.size()) processed\_InstanceID = 0;

int cnum = (database.getDataObject(Integer.toString(processed\_InstanceID++))).getClusterLabel();

if (cnum == DataObject.NOISE)

throw new Exception();

else

return cnum;

1. **Implement the following Data structures in Java**

**a)Linked Lists b) Stacks c) Queues d) Set e) Map**

**Linked List**

program:

import java.util.\*;

public class LinkedListDemo {

public static void main(String args[]) {

// create a linked list

LinkedList ll = new LinkedList();

* add elements to the linked list ll.add("F");

ll.add("B");

ll.add("D");

ll.add("E");

ll.add("C");

ll.addLast("Z");

ll.addFirst("A"); ll.add(1, "A2");

System.*out*.println("Original contents of ll: " + ll);

* remove elements from the linked list ll.remove("F");

ll.remove(2);

System.*out*.println("Contents of ll after deletion: " + ll);

* remove first and last elements ll.removeFirst(); ll.removeLast();

System.*out*.println("ll after deleting first and last: "+ ll);

* get and set a value Object val = ll.get(2);

ll.set(2, (String) val + " Changed"); System.*out*.println("ll after change: " + ll);

}

}

Output:

Original contents of ll: [A, A2, F, B, D, E, C, Z]

Contents of ll after deletion: [A, A2, D, E, C, Z]

ll after deleting first and last: [A2, D, E, C]

ll after change: [A2, D, E Changed, C]

**b) Stacks**

import java.util.\*;

public class StackDemo {

static void showpush(Stack st, int a) {

st.push(new Integer(a));

System.out.println("push(" + a + ")");

System.out.println("stack: " + st);

}

static void showpop(Stack st) {

System.out.print("pop -> ");

Integer a = (Integer) st.pop();

System.out.println(a);

System.out.println("stack: " + st);

}

public static void main(String args[]) {

Stack st = new Stack();

System.out.println("stack: " + st);

showpush(st, 42);

showpush(st, 66);

showpush(st, 99);

showpop(st);

showpop(st);

showpop(st);

try {

showpop(st);

}catch (EmptyStackException e) {

System.out.println("empty stack");

}

}

}

output:

stack: [ ]

push(42)

stack: [42]

push(66)

stack: [42, 66]

push(99)

stack: [42, 66, 99]

pop -> 99

stack: [42, 66]

pop -> 66

stack: [42]

pop -> 42

stack: [ ]

pop -> empty stack

**c) Queues**

* import java.util.LinkedList;

import java.util.Queue; public class QueueExample

{

public static void main(String[] args)

{

Queue<Integer> q = new LinkedList<>();

* + Adds elements {0, 1, 2, 3, 4} to queue

for (int i=0; i<5; i++)

q.add(i);

* Display contents of the queue. System.out.println("Elements of queue-"+q);
  + To remove the head of queue.

int removedele = q.remove(); System.out.println("removed element-" + removedele); System.out.println(q);

* To view the head of queue int head = q.peek();
* Rest all methods of collection interface,
* Like size and contains can be used with this
* implementation.

int size = q.size();

System.out.println("Size of queue-" + size);

}

}

Output:

Elements of queue-[0, 1, 2, 3, 4]

removed element-0

[1, 2, 3, 4]

head of queue-1

Size of queue-4

**d) Set**

public class SetDemo {

public static void main(String args[]) {

int count[] = {34, 22,10,60,30,22};

Set<Integer> set = new HashSet<Integer>();

try{

for(int i = 0; i<5; i++){

set.add(count[i]);

}

System.*out*.println(set);

TreeSet sortedSet = new TreeSet<Integer>(set);

System.*out*.println("The sorted list is:");

System.*out*.println(sortedSet);

System.*out*.println("The First element of the set is: "+(Integer)sortedSet.first());

System.*out*.println("The last element of the set is: "+(Integer)sortedSet.last());

}

catch(Exception e){}

}

}

Output:

[34, 22, 10, 60, 30]

The sorted list is:

[10, 22, 30, 34, 60]

The First element of the set is: 10

The last element of the set is: 60

**e) Map**

Program:

import java.awt.Color;

import java.util.HashMap;

import java.util.Map;

import java.util.Set;

public class MapDemo

{

public static void main(String[] args)

{

Map<String, Color> favoriteColors = new HashMap<String, Color>(); favoriteColors.put("sai", Color.*BLUE*); favoriteColors.put("Ram", Color.*GREEN*); favoriteColors.put("krishna", Color.*RED*);

favoriteColors.put("narayana", Color.*BLUE*); // Print all keys and values in the map

Set<String> keySet = favoriteColors.keySet(); for (String key : keySet)

{

Color value = favoriteColors.get(key);

System.*out*.println(key + " : " + value);

}

}

}

Output:

narayana : java.awt.Color[r=0,g=0,b=255]

sai : java.awt.Color[r=0,g=0,b=255]

krishna : java.awt.Color[r=255,g=0,b=0]

Ram : java.awt.Color[r=0,g=255,b=0]

**7.Aim:Study and configure hadoop for big data**

## Step 1 — Installing Java

To get started, we'll update our package list:

* sudo apt-get update

Next, we'll install OpenJDK, the default Java Development Kit on Ubuntu 16.04.

* sudo apt-get install default-jdk

Once the installation is complete, let's check the version.

* java -version

## Step 2 — Installing Hadoop

On the server, we'll use wget to fetch it:

* wget http://apache.mirrors.tds.net/hadoop/common/hadoop-2.7.3/hadoop-2.7.3.tar.gz

Again, we'll right-click to copy the file location, then use wget to transfer the file:

* wget https://dist.apache.org/repos/dist/release/hadoop/common/hadoop-2.7.3/hadoop-2.7.3.tar.gz.mds

Then run the verification:

* shasum -a 256 hadoop-2.7.3.tar.gz

Compare this value with the SHA-256 value in the .mds file:

* cat hadoop-2.7.3.tar.gz.mds
* tar -xzvf hadoop-2.7.3.tar.gz

Finally, we'll move the extracted files into /usr/local, the appropriate place for locally installed software. Change the version number, if needed, to match the version you downloaded.

* sudo mv hadoop-2.7.3 /usr/local/hadoop

With the software in place, we're ready to configure its environment.

## Step 3 — Configuring Hadoop's Java Home

To find the default Java path

* readlink -f /usr/bin/java | sed "s:bin/java::"

To begin, open hadoop-env.sh:

* sudo nano /usr/local/hadoop/etc/hadoop/hadoop-env.sh

Then, choose one of the following options:

### Option 1: Set a Static Value

#export JAVA\_HOME=${JAVA\_HOME}

export JAVA\_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre/

. . .

### Option 2: Use Readlink to Set the Value Dynamically

#export JAVA\_HOME=${JAVA\_HOME}

export JAVA\_HOME=$(readlink -f /usr/bin/java | sed "s:bin/java::")

## Step 4 — Running Hadoop

Now we should be able to run Hadoop:

* $ /usr/local/hadoop/bin/hadoop
* $ mkdir ~/input
* $ cp /usr/local/hadoop/etc/hadoop/\*.xml ~/input

**8.Aim: Hadoop commands**

Using the command line interface

In this part, we will explore some basic HDFS commands. All HDFS commands start with ***hadoop*** followed by ***dfs*** (distributed file system) or ***fs*** (file system) followed by a dash, and the command. Many HDFS commands are similar to UNIX commands. For details, refer to the *Hadoop Command Guide* and *Hadoop FS Shell Guide*.

We will start with the **hadoop fs** **–ls** command which returns the list of files and directories with permission information.

Ensure the Hadoop components are all started, and from the same Gnome terminal window as before (and logged on as *biadmin*), follow these instructions:

1. List the contents of the root directory.



**hadoop fs -ls /**

****

1. To list the contents of the /user/biadmin directory, execute:



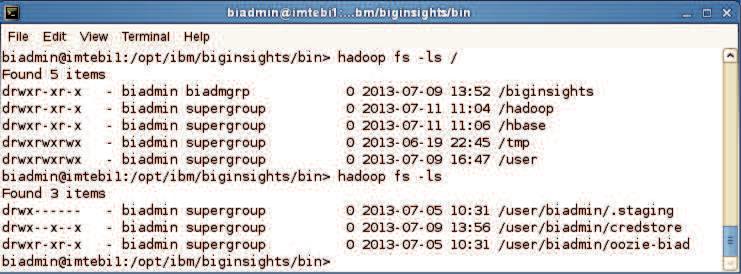
**hadoop fs -ls**

or



**hadoop fs -ls /user/biadmin**

Note that in the first command there was no directory referenced, but it is equivalent to the second command where /user/biadmin is explicitly specified. Each user will get its own home directory under /user. For example, in the case of user biadmin, his home directory is /user/biadmin. Any command where there is no explicit directory specified will be relative to the user’s home directory.



1. To create the directory ***myTestDir*** you can issue the following command:



**hadoop fs -mkdir myTestDir**

Where was this directory created? As mentioned in the previous step, any relative paths will be using the user’s home directory.

1. Issue the ls command again to see the subdirectory myTestDir:

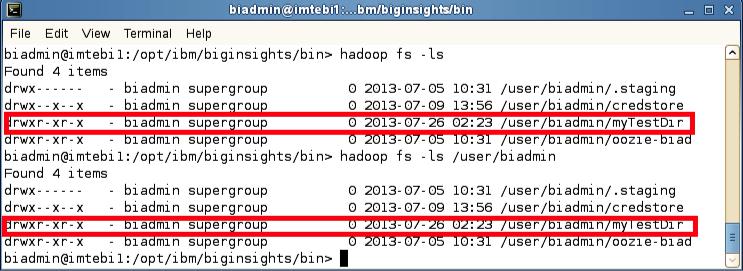


**hadoop fs -ls**

or



**hadoop fs -ls /user/biadmin**

****

To use HDFS commands recursively generally you add an “r” to the HDFS command (In the Linux shell this is generally done with the “-R” argument).

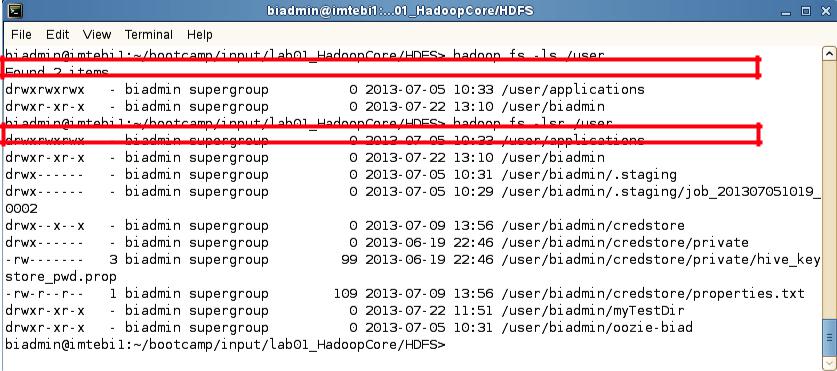
1. For example, to do a recursive listing we’ll use the –lsr command rather than just –ls, like the examples below:



**hadoop fs -ls /user**

****

**hadoop fs -lsr /user**

****

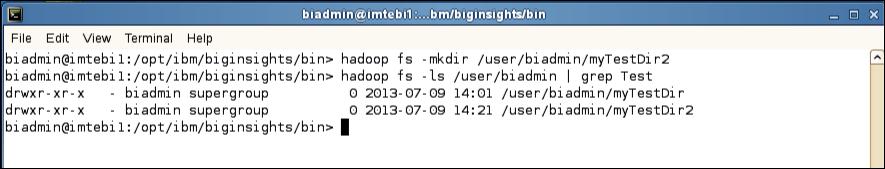
1. You can pipe (using the | character) any HDFS command to be used with the Linux shell. For example, you can easily use *grep* with HDFS by doing the following:



**hadoop fs -mkdir /user/biadmin/myTestDir2**

****

**hadoop fs -ls /user/biadmin | grep Test**

****

As you can see the grep command only returned the lines which had test in them (thus removing the “Found x items” line and the .staging and oozie-biad directories from the listing

1. To move files between your regular Linux filesystem and HDFS you can use the put and get commands. For example, move the text file README to the hadoop filesystem.



**hadoop fs -put /home/biadmin/bootcamp/input/lab01\_HadoopCore/HDFS/README**

****

**README**

****

**hadoop fs -ls /user/biadmin**

****

You should now see a new file called /user/biadmin/README listed as shown above. Note there is a ‘1’ highlighted in the figure. This represents the replication factor. By default, the replication factor in a BigInsights cluster is 3, but since this laboratory environment only has one node, the replication factor is 1.

1. In order to view the contents of this file use the –cat command as follows:



**hadoop fs -cat README**

You should see the output of the README file (that is stored in HDFS). We can also use the linux diff command to see if the file we put on HDFS is actually the same as the original on the local filesystem.

1. Execute the commands below to use the diff command:



**cd /home/biadmin/bootcamp/input/lab01\_HadoopCore/HDFS/**

****

**diff <( hadoop fs -cat README ) README**

Since the diff command produces no output we know that the files are the same (the diff command prints all the lines in the files that differ).

To find the size of files you need to use the –du or –dus commands. Keep in mind that these commands return the file size in bytes.

10. To find the size of the README file use the following command:



**hadoop fs -du README**

****

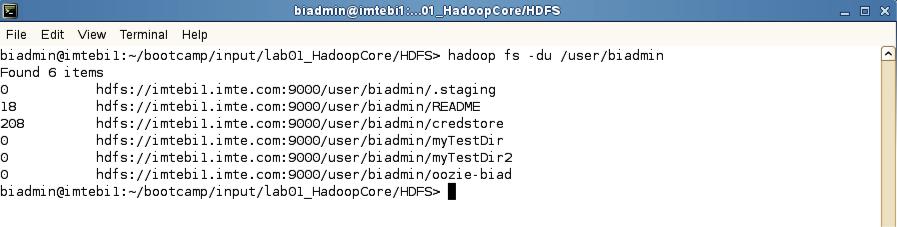
**Inspecting README file size**

In this example, the README file has 18 bytes.

11. To find the size of all files individually in the /user/biadmin directory use the following command:

****

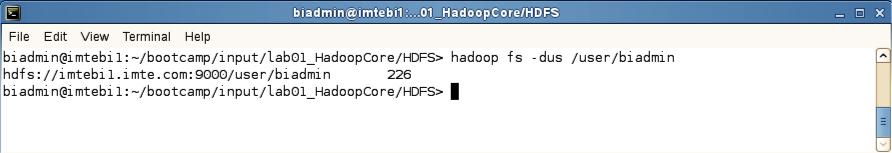
**hadoop fs -du /user/biadmin**

****

12. To find the size of all files in total of the /user/biadmin directory use the following command:



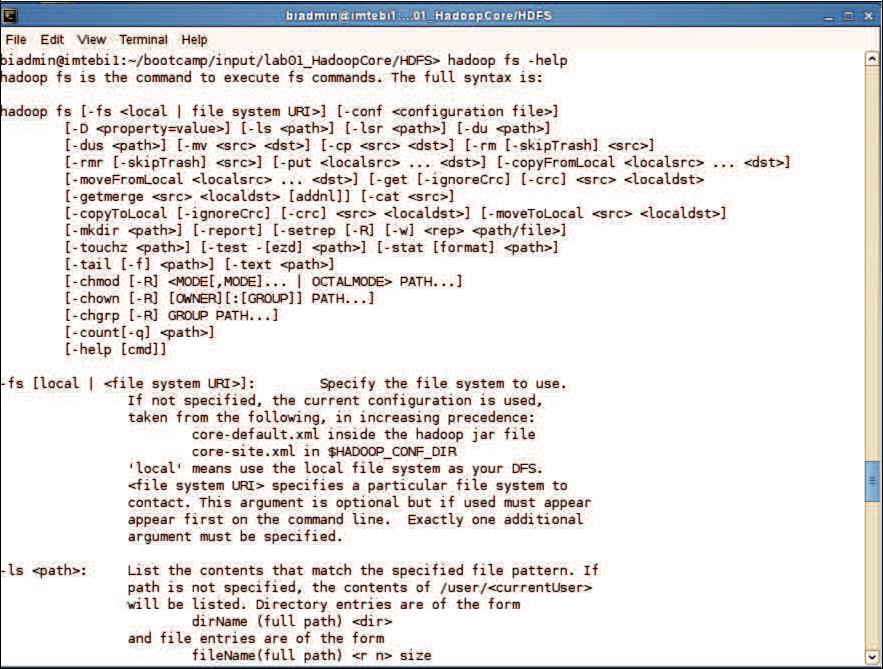
**hadoop fs -dus /user/biadmin**

****

13. If you would like to get more information about hadoop fs commands, invoke –help as follows:



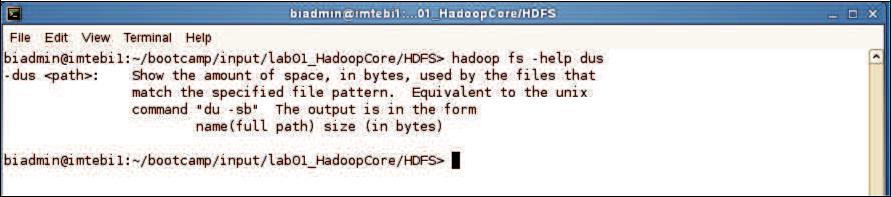
**hadoop fs -help**

****

1. For specific help on a command, add the command name after help. For example, to get help on the dus command you’d do the following:



**hadoop fs -help dus**

****

**Help for specific Haoop commands**

**9.Aim: Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.**

Now that we’ve seen how the FileSystem (fs) shell can be used to execute Hadoop commands to interact with HDFS, the same fs shell can be used to launch MapReduce jobs. In this section, we will walk through the steps required to run a MapReduce program. The source code for a MapReduce program is contained in a compiled .jar file. Hadoop will load the JAR into HDFS and distribute it to the data nodes, where the individual tasks of the MapReduce job will be executed. Hadoop ships with some example MapReduce programs to run. One of these is a distributed WordCount program which reads text files and counts how often words occur.

**Running the WordCount program**

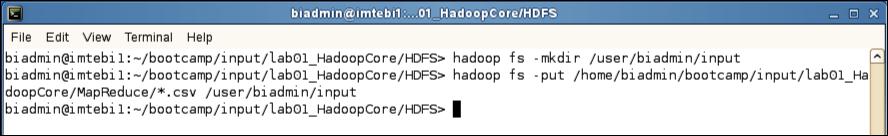
First we need to copy the data files from the local file system to HDFS.

Step 1:Execute the commands below to copy the input files into HDFS.



**hadoop fs -mkdir /user/biadmin/input**

**hadoop fs -put /home/biadmin/bootcamp/input/lab01\_HadoopCore/MapReduce/\*.csv /user/biadmin/input**

****

**Copy input files into HDFS**

Step 2: Review the files have been copied with the following command:



**hadoop fs -ls input**

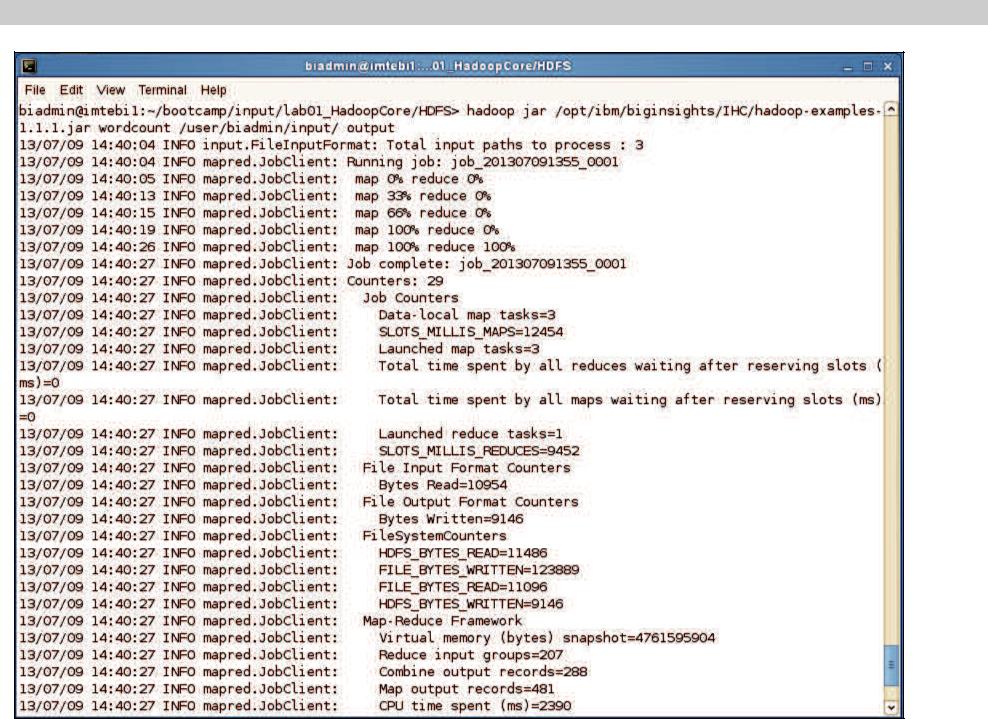
****

**List copied files into HDFS**

Step 3: Now we can run the wordcount job with the command below, where “/user/biadmin/input/” is where the input files are, and “output” is the directory where the output of the job will be stored. The “output” directory will be created automatically when executing the command below.

****

**hadoop jar /opt/ibm/biginsights/IHC/hadoop-examples-1.1.1.jar wordcount /user/biadmin/input/ output**

****

**WordCount MapReduce job running**

Step 4: Now review the output of step 3:

In this case, the output was not split into multiple files.



**hadoop fs -ls output**

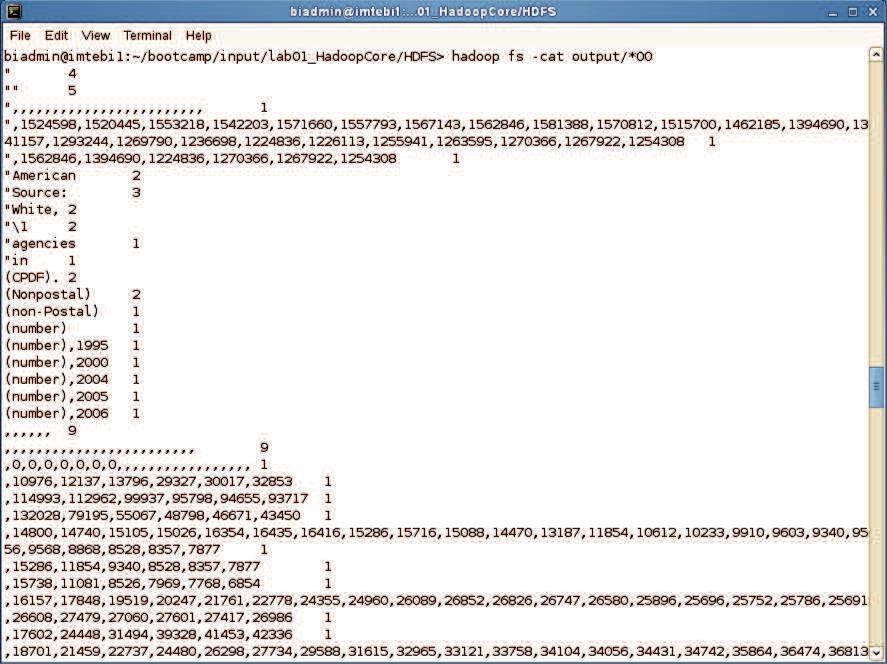
****

**MapReduce result files**

Step 5: To view the contents of the part-r-0000 file issue the command below:



**hadoop fs -cat output/\*00**

****

**MapReduce output**

1. **Implement Matrix Multiplication with Hadoop Map Reduce**

