Instructions For Running Map Reduce Program

**How to log in by SSH terminal?**

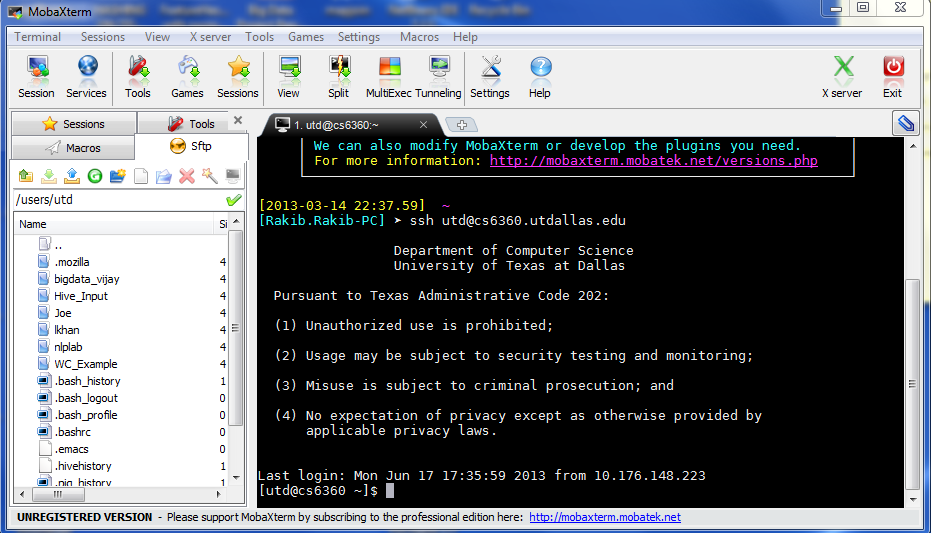
Please download an SSH client on your Windows laptops before you arrive to the workshop.  Here are the download URL's:  
  
      <http://mobaxterm.mobatek.net/MobaXterm_Setup_6.3.msi>  
      <http://www.hlt.utdallas.edu/MobaXterm_Setup_6.3.msi>

Log in **cs6360.utdallas.edu** with

Username: **utd**

Password: **hadoop.**

Run **MobaXterm** and type **utd@cs6360.utdallas.edu**

****

**How to Compile the .java file and build .jar file?**

You can compile java file and also build jar file by using any standard IDE like Eclipse / NetBeans or any stand java IDE.

On the other hand you can compile java file and then create jar file from command line on Putty terminal

Step 1:  **javac -classpath <hadoop-core-1.0.4.jar file path> <java file path>**

Step 2: **jar cvf <jar file> -C <manifest file directory> <class file directory>**

**Example: Word Count**

In Word Count example using Map Reduce,  counts the appearance of each word in a set of documents.

You will see **WC\_Example** in your parent directory **/users/utd/** and you will find files **file01**, **file02** inside **WC\_Example**. Your task is to count the frequency of each word for these two files.

1. Make input folder in HDFS and copy file01 and file02

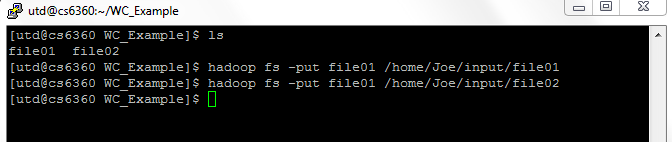
**hadoop fs -put file01 /home/<your\_name>/input/file01**

**hadoop fs -put file01 /home/<your\_name>/input/file02**

e.g.

**hadoop fs -put file01 /home/Joe/input/file01**

**hadoop fs -put file01 /home/Joe/input/file02**

****

2. Go to your directory (Joe) and create a folder **WC**.

3. Inside WC folder copy/upload **WordCount.java** by WinSCP

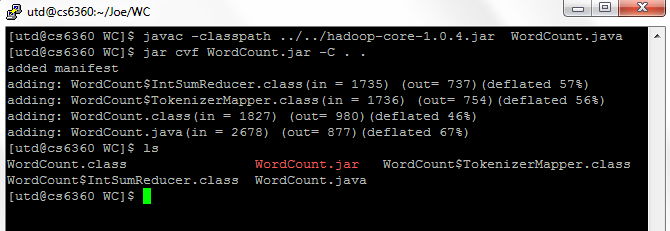
Source code of WordCount.java can be found later.

4. Run the following commands:

**javac -classpath ../../hadoop-core-1.0.4.jar WordCount.java**

**jar cvf WordCount.jar -C . .**

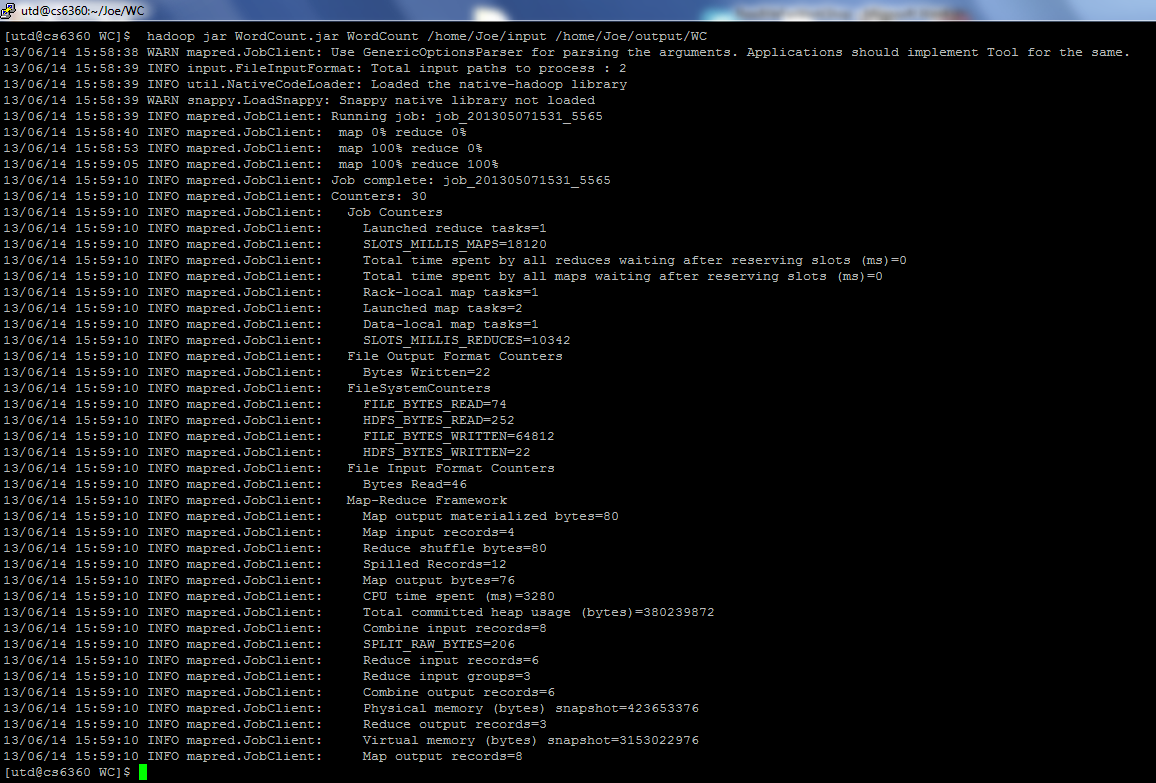
4. You will see a jar file named WordCount.jar is created in WC folder.



5. Run map reduce program

hadoop jar WordCount.jar WordCount /home/Joe/input /home/Joe/output/WC

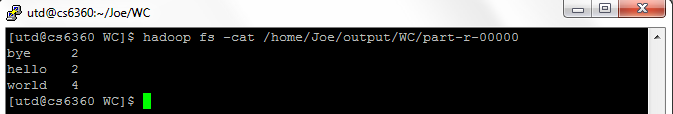
This WordCount program takes the input files location (/home/Joe/input) and output file location (/home/Joe/output/WC).



6. You will see the **part-r-00000** file in **/home/Joe/output/WC** directory. The following command will give you the output.

**hadoop fs -cat /home/Joe/output/WC/part-r-00000**

This will give the following output.



**Hands on Exercise**

In this exercise you will learn how to solve problems using Map Reduce.

Please apply Hadoop mapreduce to derive some statistics from White House Visitor Log. There are currently 2.9 million records available at

<http://www.whitehouse.gov/briefing-room/disclosures/visitor-records>

Data is available as web only spreadsheet view and downloadable raw format in CSV (Comma Separated Value). In CSV format each column is separated by a comma “,” in each line. The first line represents the heading for the corresponding columns in other lines. We are going to use this raw data for our mapreduce operation.

Download the 4 CSV files (zipped) from the site (at the bottom of the page), unzip, and copy to the UTD’s HDFS directory - “/home/<your\_utd\_netid>/whitehouse/input”. Use the *put* or *copyFromLocal* HDFS shell command to copy those files. Also produce the output to “/home/<your\_name>/output” HDFS directory. For more information about HDFS shell, see here -

<http://hadoop.apache.org/docs/stable/file_system_shell.html>

You are required to write efficient Hadoop ***MapReduce programs in Java*** to find the following information:

**N.B.** The customized dataset will be found at **/WH/input** and you should use this as input.

You are required to write efficient Hadoop ***MapReduce programs in Java*** to find the following information:

*(i) The 10 most frequent visitors (NAMELAST, NAMEFIRST) to the White House.*

*(ii) The 10 most frequently visited people (visitee\_namelast, visitee\_namefirst) in the White House.*

*(iii) The 10 most frequent visitor-visitee combinations.*

Consider mapreduce chaining for efficiency. See the possible ways for chaining in Yahoo’s tutorial and the stackoverflow discussion

<http://developer.yahoo.com/hadoop/tutorial/module4.html#chaining>

<http://stackoverflow.com/questions/3059736/map-reduce-chainmapper-and-chainreducer>

**(i) The 10 most frequent visitors (NAMELAST, NAMEFIRST) to the White House.**

1. Create a folder WH inside folder Joe(folder created earlier).

**mkdir WH**

2. Inside WH folder copy/upload WhiteHouse.java by WinSCP

3. Run the following commands:

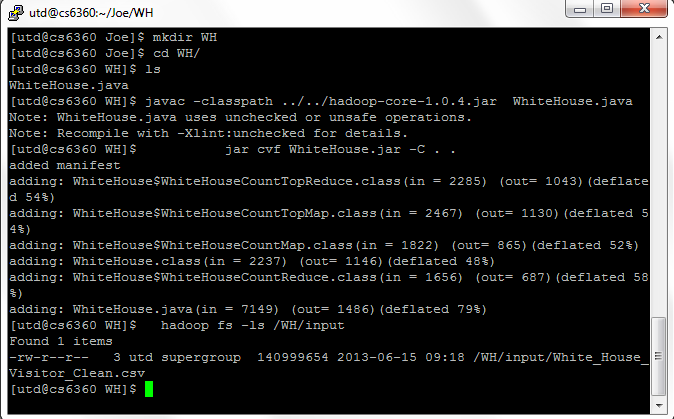
**javac -classpath ../../hadoop-core-1.0.4.jar WhiteHouse.java**

**jar cvf WhiteHouse.jar -C . .**

4. You will see a jar file named WhiteHouse.jar is created in WH folder

5. Make sure that input files are inserted proper directory.

**hadoop fs -ls /WH/input**

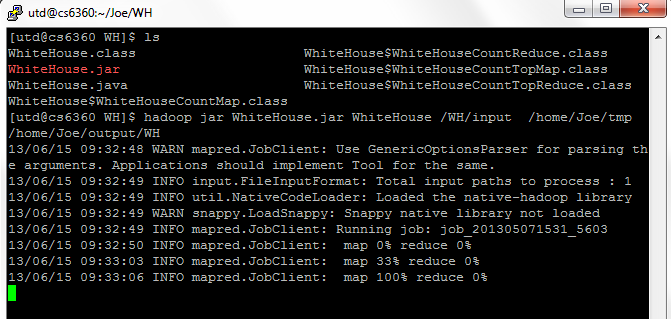
****

6. Run map reduce program

**hadoop jar WhiteHouse.jar WhiteHouse /WH/input /home/Joe/tmp /home/Joe/output/WH**

(For chaining purpose the java program requires three file locations. First is input file location, the second is the intermediate file location and the third one is the final output file location.)

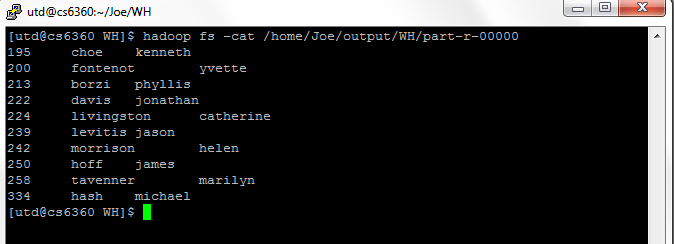
This will gives you following which indicates that map and reduce job status

****

7. You will see the **part-r-00000** file in **/home/Joe/output/WH** directory. The following command will give you the output.

**hadoop fs -cat /home/Joe/output/WH/part-r-00000**

This will give the following output.



**WordCount.java**

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.fs.FileSystem;

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;

import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

public static class TokenizerMapper

extends Mapper<Object, Text, Text, IntWritable>{

private final static IntWritable one = new IntWritable(1);

private Text word = new Text();

@Override

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<Text,IntWritable,Text,IntWritable> {

private IntWritable result = new IntWritable();

/\*\*

\*

\* @param key

\* @param values

\* @param context

\* @throws IOException

\* @throws InterruptedException

\*/

@Override

public void reduce(Text key, Iterable<IntWritable> 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();

if (args.length != 2) {

System.err.println("Usage: WordCount <in> <out>");

System.exit(2);

}

FileSystem fs = FileSystem.get(conf);

fs.delete(new Path(args[1]), true);

Job job = new Job(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);

}

}

**WhiteHouse.java**

import java.io.IOException;

import java.util.TreeMap;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.NullWritable;

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;

/\*\*

\*

\* @author Rakib

\*/

public class WhiteHouse {

public static class WhiteHouseCountMap extends Mapper<Object, Text, Text, IntWritable>

{

private final static IntWritable one = new IntWritable(1);

private Text word = new Text();

@Override

public void map(Object key, Text value, org.apache.hadoop.mapreduce.Mapper.Context context) throws IOException, InterruptedException {

String[] container = value.toString().split(",");

//StringTokenizer tokenizer = new StringTokenizer(line, ",");

if (container.length > 0) {

String firstName = container[1].trim();

String lastName = container[0].trim();

if (!"NAMELAST".equals(lastName) && !"NAMEFIRST".equals(firstName)) {

String name = lastName + "\t" + firstName;

word.set(name);

context.write(word, one);

}

}

}

}

public static class WhiteHouseCountReduce extends Reducer<Text, IntWritable, Text, IntWritable> {

@Override

public void reduce(Text key, Iterable<IntWritable> values, Context context

) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

context.write(key, new IntWritable(sum));

}

}

public static class WhiteHouseCountTopMap extends Mapper<Object, Text, NullWritable, Text> {

private TreeMap<Integer, Text> visitorToRecordMap = new TreeMap<Integer, Text>();

@Override

public void map(Object key, Text value, org.apache.hadoop.mapreduce.Mapper.Context context) throws IOException, InterruptedException {

String[] container = value.toString().split("\t");

if (container.length > 0)

{

String containerValue = container[2].trim() + "\t" + container[0].trim() + "\t" + container[1].trim();

visitorToRecordMap.put(Integer.parseInt(container[2].trim()), new Text(containerValue));

if (visitorToRecordMap.size() > 10)

{

visitorToRecordMap.remove(visitorToRecordMap.firstKey());

}

}

}

@Override

protected void cleanup(org.apache.hadoop.mapreduce.Mapper.Context context) throws IOException,

InterruptedException {

// Output our ten records to the reducers with a null key

for (Text t : visitorToRecordMap.values()) {

context.write(NullWritable.get(), t);

}

}

}

public static class WhiteHouseCountTopReduce extends Reducer<NullWritable, Text, NullWritable, Text> {

private TreeMap<Integer, Text> visitorToRecordMap = new TreeMap<Integer, Text>();

public void reduce(NullWritable key, Iterable<Text> values, Reducer.Context context

) throws IOException, InterruptedException {

for (Text val : values)

{

String value = val.toString();

String[] container = value.split("\t");

//StringTokenizer tokenizer = new StringTokenizer(line, ",");

if (container.length > 0)

{

String count = container[0].trim();

visitorToRecordMap.put(Integer.parseInt(count), new Text(value));

if (visitorToRecordMap.size() > 10)

{

visitorToRecordMap.remove(visitorToRecordMap.firstKey());

}

}

}

for (Text t : visitorToRecordMap.values()) {

//System.out.println(t.toString());

context.write(NullWritable.get(), t);

}

}

}

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

if (args.length != 3) {

System.err.println("Usage: whitehouse <in> <temp> <out>");

System.exit(2);

}

FileSystem fs = FileSystem.get(conf);

fs.delete(new Path(args[1]), true);

fs.delete(new Path(args[2]), true);

Job countingJob = new Job(conf, "whitehouse\_count");

countingJob.setJarByClass(WhiteHouse.class);

countingJob.setMapperClass(WhiteHouseCountMap.class);

countingJob.setCombinerClass(WhiteHouseCountReduce.class);

countingJob.setReducerClass(WhiteHouseCountReduce.class);

countingJob.setOutputKeyClass(Text.class);

countingJob.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(countingJob, new Path(args[0]));

FileOutputFormat.setOutputPath(countingJob, new Path(args[1]));

if (countingJob.waitForCompletion(true))

{

Job countingTopJob = new Job(conf, "whitehouse\_top");

countingTopJob.setJarByClass(WhiteHouse.class);

countingTopJob.setMapperClass(WhiteHouseCountTopMap.class);

countingTopJob.setCombinerClass(WhiteHouseCountTopReduce.class);

countingTopJob.setReducerClass(WhiteHouseCountTopReduce.class);

countingTopJob.setOutputKeyClass(NullWritable.class);

countingTopJob.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(countingTopJob, new Path(args[1]));

FileOutputFormat.setOutputPath(countingTopJob, new Path(args[2]));

System.exit(countingTopJob.waitForCompletion(true) ? 0 : 1);

}

}

}

Pig Latin Hands on Exercise

**Dataset**:

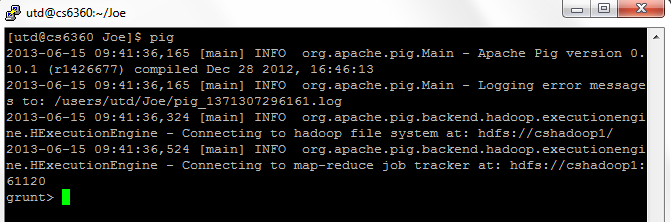
We will use the White House datasets located under **/Pig/top10/input** in the HDFS in the Programming/Master Node CS6360.utdallas.edu. Please use this folder and don’t copy to any other folder on the server. All datasets are comma separated and each line has the following 11 columns NAMELAST,NAMEFIRST,NAMEMID,UIN,BDGNBR,ACCESS\_TYPE,TOA,POA,TOD,POD,APPT\_MADE\_DATE.

**Requirement:**

Using Pig Latin commands, find the 10 most frequent visitors (NAMELAST, NAMEFIRST) to the White House.

1. Log in cs6360.utdallas.edu and Run pig

* **pig**



The above dialog shows the interactive mode. In this mode you can execute pig commands one by one.

2. Run the following commands sequentially.

* Load the input CSV file into a variable A

**A = load '/Pig/top10/input' using PigStorage(',') as (NAMELAST,NAMEFIRST,NAMEMID,UIN,BDGNBR,ACCESS\_TYPE,TOA,POA,TOD,POD,APPT\_MADE\_DATE);**

* Group A by NAMELAST,NAMEFIRST and put it in to variable B

**B = group A by (NAMELAST,NAMEFIRST);**

* For each group find the number of visits and put it to variable C

**C = foreach B generate group, COUNT(A.(NAMELAST,NAMEFIRST)) as num\_of\_visits;**

* Sort visitors by descending order with their number of visits and put these to variable D.

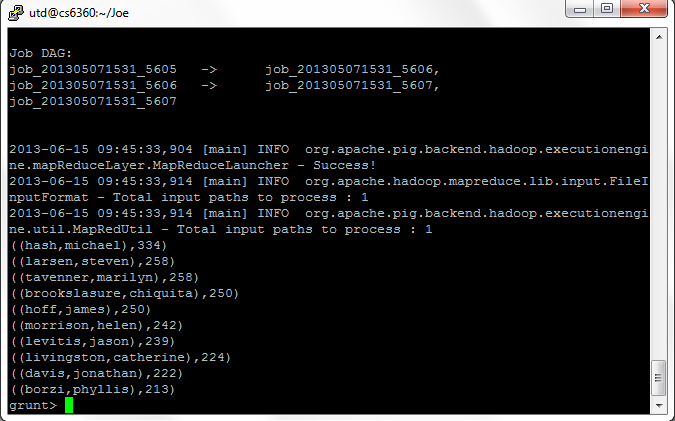
**D = order C by num\_of\_visits desc;**

* Take the first 10 visitors.

**E = limit D 10;**

* See the output

**dump E;**

****

* Store the result to a file.

**store E into '/home/Joe/Pig/output/top10/top\_visitors';**

The file is stored at HDFS. The file can be viewed by

**hadoop fs -cat /home/Joe/Pig/output/top10/top\_visitors/part-r-00000**

**Datasets**:

The three datasets **( /Pig/join/input**) that will be used are as follows:

* NASA\_HTTP.txt: The delimiter is tab and each line has the following 2 columns IP, VALUE.
* HOST\_COUNTRY.txt: The delimiter is tab and each line has the following 2 columns IP, COUNTRY ABBREVIATION.
* COUNTRY\_NAME.txt: The delimiter is tab and each line has the following 2 columns COUNTRY ABBREVIATION, COUNTRY NAME

**Requirement**:

1. Write Pig Latin commands to do multiple tables inner join for the above mentioned datasets (***the join attribute is (IP) for the first two datasets and country abbreviation for the second and third datasets***.)

Load the input text file (tab delimited) file into a variable A, B and C

* **A = load '/Pig/join/input/NASA\_HTTP.txt' using PigStorage('\t') as (IP,VALUE);**
* **B = load '/Pig/join/input/HOST\_COUNTRY.txt' using PigStorage('\t') as (IP, COUNTRY\_ABBREVIATION);**
* **C= load '/Pig/join/input/COUNTRY\_NAME.txt' using PigStorage('\t') as (COUNTRY\_ABBREVIATION, COUNTRY\_NAME);**

Join A and B by IP and put the result into D

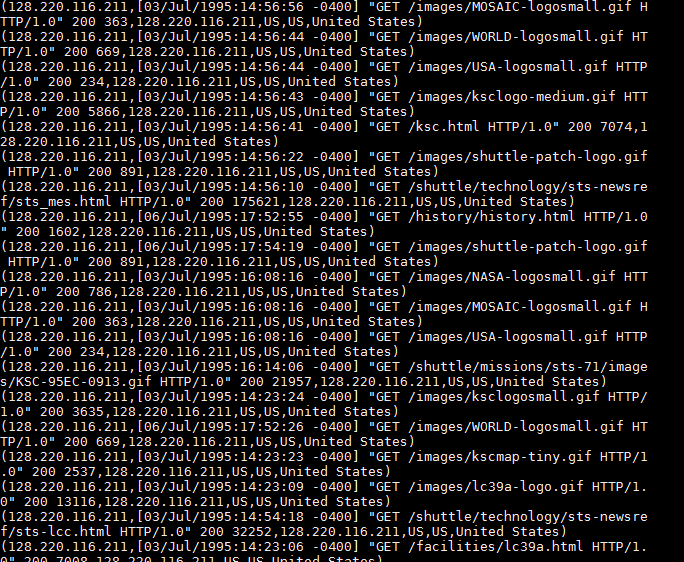
* **D = join A by IP, B by IP;**

Join D and C by **COUNTRY\_ABBREVIATION** and put the result into E

* **E = join D by COUNTRY\_ABBREVIATION, C by COUNTRY\_ABBREVIATION;**

Show the output

* **dump E;**

****

* **store E into '/home/Joe/Pig/output/join/nasa';**

The file is stored at HDFS. The file can be viewed by

**hadoop fs -cat /home/Joe/Pig/output/join/nasa/part-r-00000**

2. Implement Co-group command on IP for the datasets NASA\_HTTP and HOST\_COUNTRY

Load the input text file (tab delimited) file into a variable A, B and C

* **A = load '/Pig/join/input/NASA\_HTTP.txt' using PigStorage('\t') as (IP,VALUE);**
* **B = load '/Pig/join/input/HOST\_COUNTRY.txt' using PigStorage('\t') as (IP, COUNTRY\_ABBREVIATION);**

Co-group A and B by IP and put the result into D

* **C = cogroup A by IP, B by IP;**

Show the output

* **dump C;**
* **store C into '/home/Joe/Pig/output/cogroup/nasa';**

The file is stored at HDFS. The file can be viewed by

**hadoop fs -cat /home/Joe/Pig/output/cogroup/nasa/part-r-00000**

****

Hive Hands on Exercise

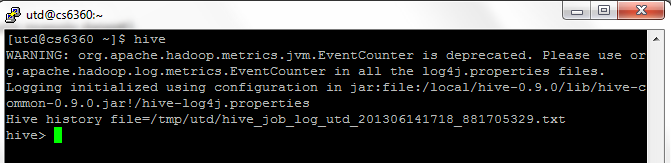
**Dataset**:

We will use the White House datasets located under **/users/utd/Hive\_Input/top10** in the HDFS in the Programming/Master Node CS6360.utdallas.edu. Please use this folder and don’t copy to any other folder on the server. All datasets are comma separated and each line has the following 11 columns NAMELAST,NAMEFIRST,NAMEMID,UIN,BDGNBR,ACCESS\_TYPE,TOA,POA,TOD,POD,APPT\_MADE\_DATE.

**Requirement:**

Using Hive commands, find the 10 most frequent visitors (NAMELAST, NAMEFIRST) to the White House (use Pig Latin example dataset).

1. Log in cs6360.utdallas.edu and Run hive



The above dialog shows the interactive mode. In this mode you can execute hive commands one by one.

2. Run the following commands sequentially.

* Create table visitor

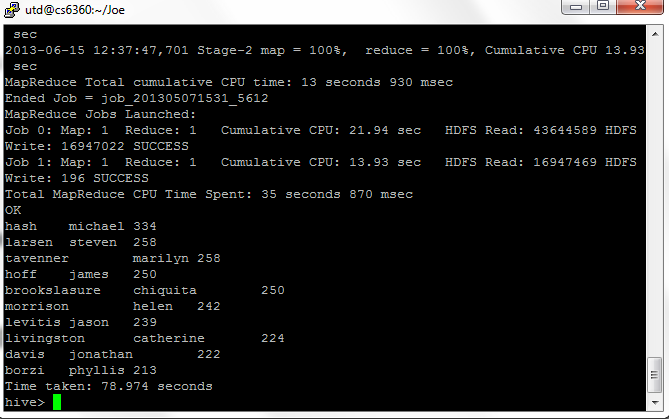
**create table visitor (NAMELAST STRING,NAMEFIRST STRING,NAMEMID STRING,UIN STRING,BDGNBR STRING,ACCESS\_TYPE STRING,TOA STRING,POA STRING,TOD STRING,POD STRING,APPT\_MADE\_DATE STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';**

* Load data from input CSV file into a table visitor

**LOAD DATA local INPATH '/users/utd/Hive\_Input/top10/White\_House.txt' into table visitor;**

* Sort visitors by descending order with their number of visits and fetch top 10 visitors

**select NAMELAST, NAMEFIRST, count(\*) as visitor\_count from visitor group by NAMELAST, NAMEFIRST order by visitor\_count desc limit 10;**



**Datasets**:

The three datasets (**/users/utd/Hive\_Input/nasa**) that will be used are as follows:

1. NASA\_HTTP.txt: The delimiter is tab and each line has the following 2 columns IP, VALUE.
2. HOST\_COUNTRY.txt: The delimiter is tab and each line has the following 2 columns IP, COUNTRY ABBREVIATION.
3. COUNTRY\_NAME.txt: The delimiter is tab and each line has the following 2 columns COUNTRY ABBREVIATION, COUNTRY NAME.

The three datasets are located under **/users/utd/Hive\_Input/nasa** in the **Local** Unix System. Please use this folder and don’t copy to any other folder on the server.

**Requirement**:

Write hive commands to do multiple tables inner join for the above mentioned datasets (***the join attribute is (IP) for the first two datasets and country abbreviation for the second and third datasets***.)

* Create table nasa and load data from NASA\_HTTP.txt

**create table nasa (IP STRING,VALUE STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';**

**LOAD DATA local INPATH '/people/cs/k/kma041000/BigData/hive/nasa/NASA\_HTTP.txt' into table nasa;**

* Create table **host** and load data from **HOST\_COUNTRY**.txt

**create table host (IP STRING,COUNTRY\_ABBREVIATION STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';**

**LOAD DATA local INPATH '/people/cs/k/kma041000/BigData/hive/nasa/HOST\_COUNTRY.txt' into table host;**

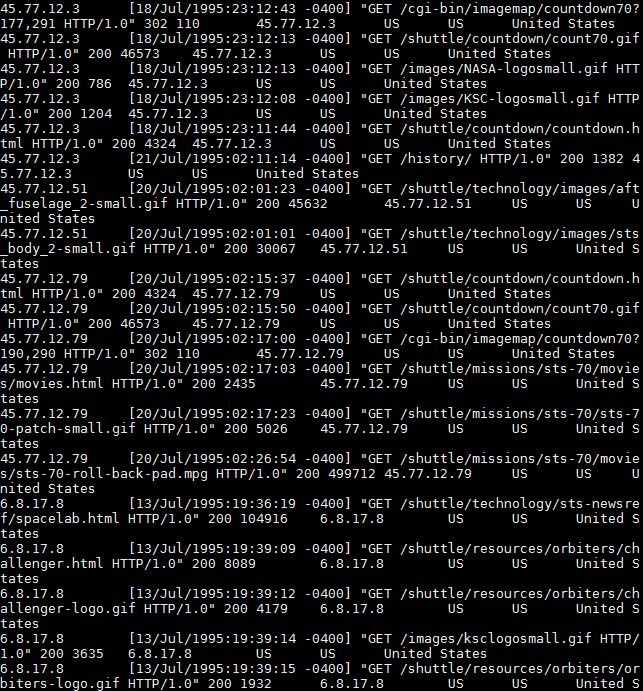
* Create table **country** and load data from **COUNTRY\_NAME**.txt

**create table country(COUNTRY\_ABBREVIATION STRING, COUNTRY\_NAME STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';**

**LOAD DATA local INPATH '/people/cs/k/kma041000/BigData/hive/nasa/COUNTRY\_NAME.txt' into table country;**

* Join **nasa** and **host** by **IP** and **country** and **host** by **country\_abbreviation**

**select \* from nasa join host on (nasa.ip = host.ip) join country on (country.country\_abbreviation = host.country\_abbreviation);**



**Hands On Exercise on Mahout**

**Clustering**

**Dataset**:

All the instances are located in the HDFS path */reviews-extracted*. Please don’t copy to any other location. Data has been extracted from the imdb62 dataset \*. These movie reviews were crawled from www.imdb.com in May 2009.

Each line in imdb62 is in the following tab-separated format:

(userId) (itemId ) (title+content)

where:

- userId is the user's ID, as used in IMDb (e.g., for userId=33913 the user's review page is http://www.imdb.com/user/ur0033913/comments)

- itemId is the item's ID. For this section, IDs have been taken starting from 1 to N where N is the number of instances.

- title is the review's title

- content is the review's content

For this section, 5000 instances have been used from this dataset. Each instance has been rewritten to a separate file (document) with a slight modification to suite Mahout Requirements.

**Requirements:**

Apply k-means clustering algorithm in Mahout considering the following:

* Go to mahout bin folder

**cd /usr/local/mahout-0.7/bin/**

* Build the sequence file from text file in a given input directory.

**./mahout seqdirectory -c UTF-8 -i /reviews-extracted/ -o /Joe/kmeans**

* For building the dictionary, eliminate any token that appears in 80% of the docs. Apply TF-IDF in Mahout to generate the vectors.

**./mahout seq2sparse -i /Joe/kmeans/ -o /Joe/kmeans-vector/ -ow -chunk 100 -x 80**

* Generate 60 clusters. For the similarity measure, use the Cosine Distance Measure.

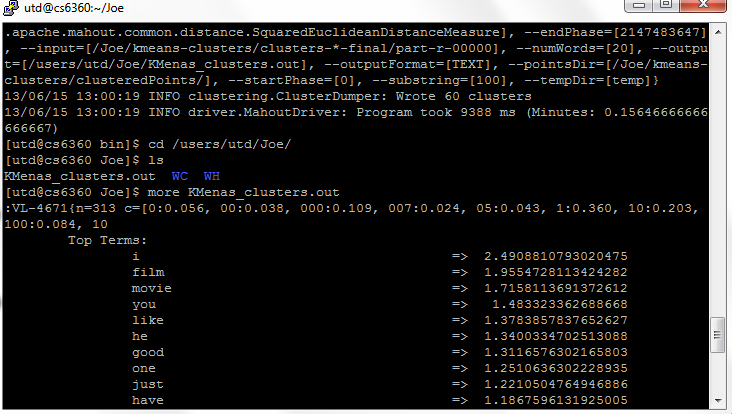
**./mahout kmeans -i /Joe/kmeans-vector/tfidf-vectors/ -c /Joe/kmeans-centroids -cl -o /Joe/kmeans-clusters -k 60 -ow -x 10 -dm org.apache.mahout.common.distance.CosineDistanceMeasure**

* Show the clusters and clustered points.

**./mahout clusterdump -d /Joe/kmeans-vector/dictionary.file-0 -dt sequencefile -i /Joe/kmeans-clusters/clusters-\*-final/part-r-00000 -n 20 -b 100 -o /users/utd/Joe/KMenas\_clusters.out -p /Joe/kmeans-clusters/clusteredPoints/**

**./mahout seqdumper -i /Joe/kmeans-clusters/clusteredPoints/part-m-00000**

You will see a file **KMenas\_clusters.out** in **/users/utd/Joe**.

****

**Classification**

**Dataset**:

The same dataset mentioned in Clustering example will be used for this question as well. The sequence files have already been generated and are located in the HDFS location /*reviews-seq*. Please don’t copy to any other location.

**Requirements**:

Apply Naïve Bayes Classification algorithm in Mahout considering the following:

* Go to mahout bin folder

**cd /usr/local/mahout-0.7/bin/**

* For building the dictionary, eliminate any token that appears in 60% of the docs. Apply TF-IDF in Mahout to generate the vectors.

**./mahout seq2sparse -i /reviews-seq -o /Joe/nb-vectors -ow -x 60**

* Use 70% for Training the model and 30% for testing.

**./mahout split -i /Joe/nb-vectors/tfidf-vectors --trainingOutput /Joe/nb-train-vectors --testOutput /Joe/nb-test-vectors --randomSelectionPct 30 --overwrite --sequenceFiles -xm sequential**

**./mahout trainnb -i /Joe/nb-train-vectors -el -li labelindex -o /Joe/nb-model -ow -c**

* Test the *Training* set to show the Correctly Classified Instances and the show Confusion Matrix.

**clear**

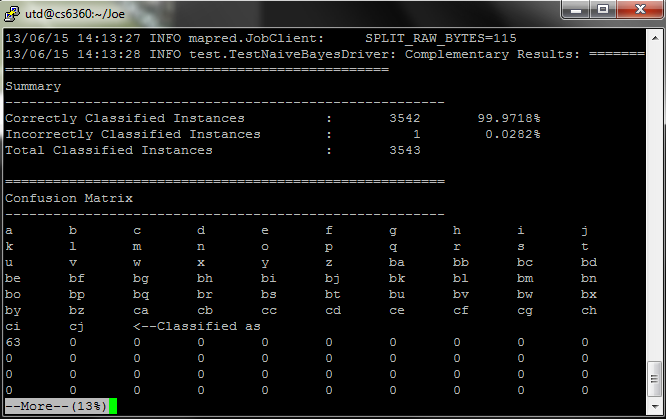
**./mahout testnb -i /Joe/nb-train-vectors -m /Joe/nb-model -l labelindex -ow -o /Joe/nb-tainingtest -c 2> /users/utd/Joe/Classifier\_TrainingModelTest.txt**

* Test the *Testing* set to show the Correctly Classified Instances and the show Confusion Matrix.

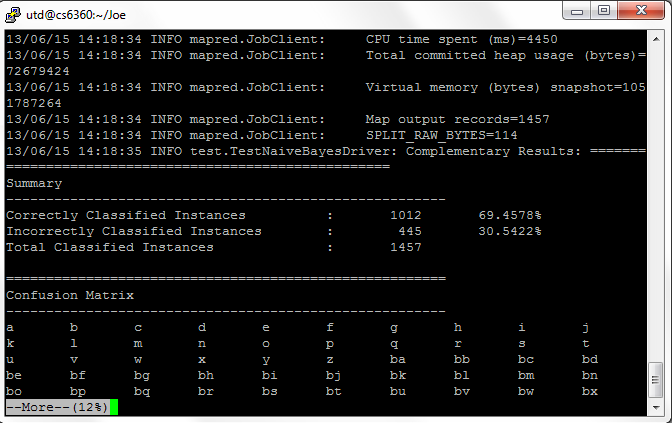
**clear**

**./mahout testnb -i /Joe/nb-test-vectors -m /Joe/nb-model -l labelindex -ow -o /Joe/nb-testingtest -c 2> /users/utd/Joe/Classifier\_TestingModelTest.txt**

Open the **/users/utd/Joe/Classifier\_TrainingModelTest.txt (**On terminal use **more /users/utd/Joe/Classifier\_TrainingModelTest.txt)**

****

Open the **/users/utd/Joe/Classifier\_TestingModelTest.txt (**On terminal use **more /users/utd/Joe/Classifier\_TestingModelTest.txt)**

****