**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:



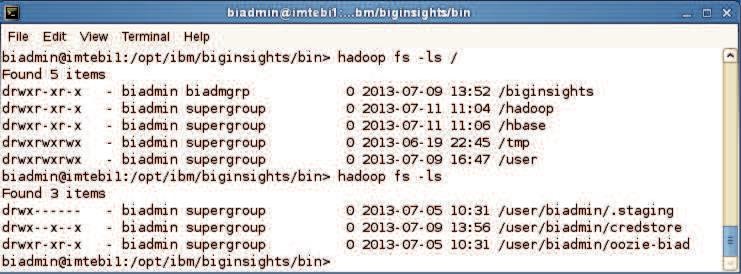
**hadoop fs -ls**

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:

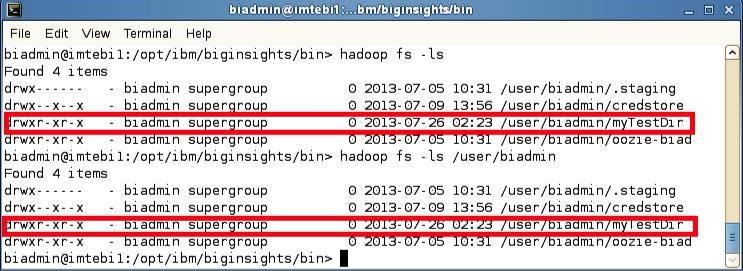


**hadoop fs -ls**

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).

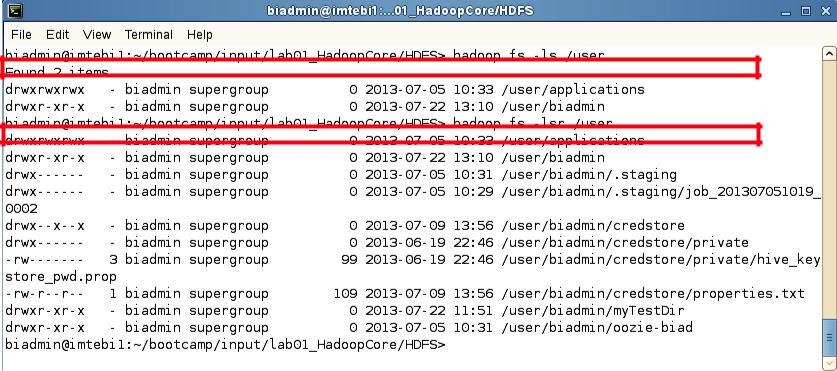
1. For example, to do a recursive listing we’ll use the –lsr command rather than just –ls, like the examples below:



**hadoop fs -ls /user**



**hadoop fs -lsr /user**



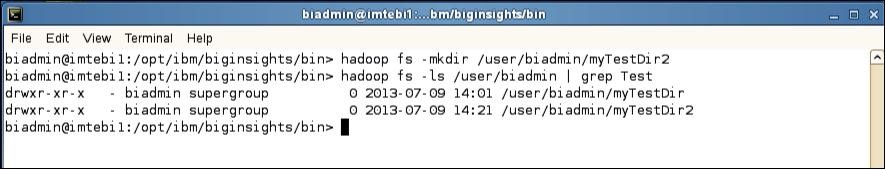
1. You can pipe (using the | character) any HDFS command to be used with the Linux shell. For example, you can easily use *grep* with HDFS by doing the following:



**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**



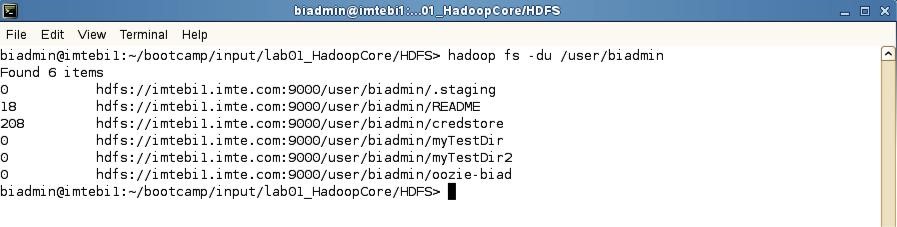
**Inspecting README file size**

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:



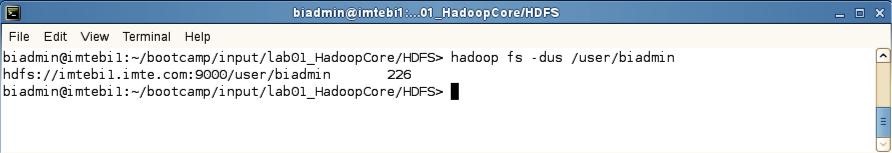
**hadoop fs -du /user/biadmin**



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



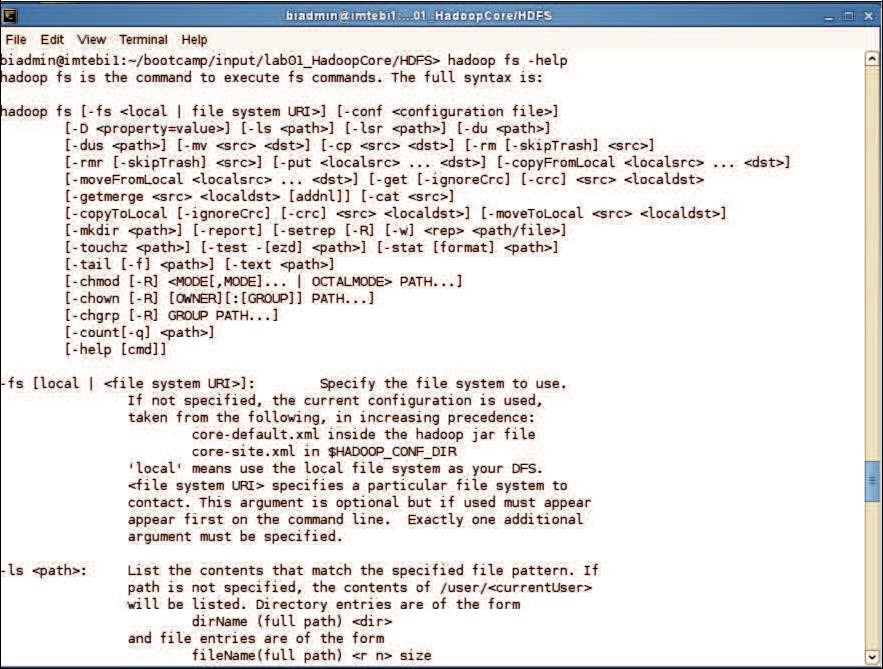
**hadoop fs -dus /user/biadmin**



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



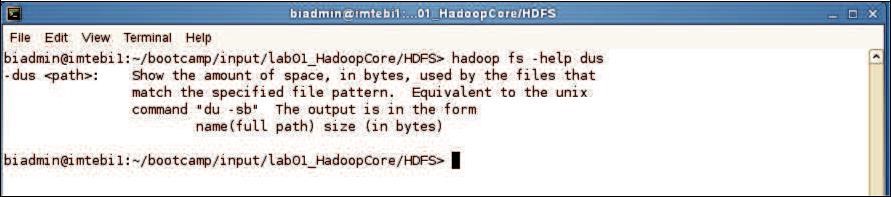
**hadoop fs -help**



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**

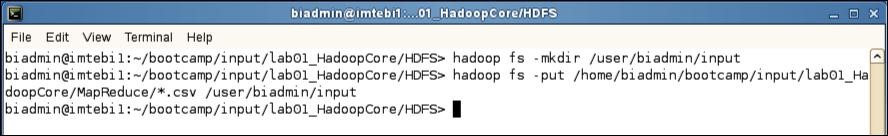
First we need to copy the data files from the local file system to HDFS.

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:



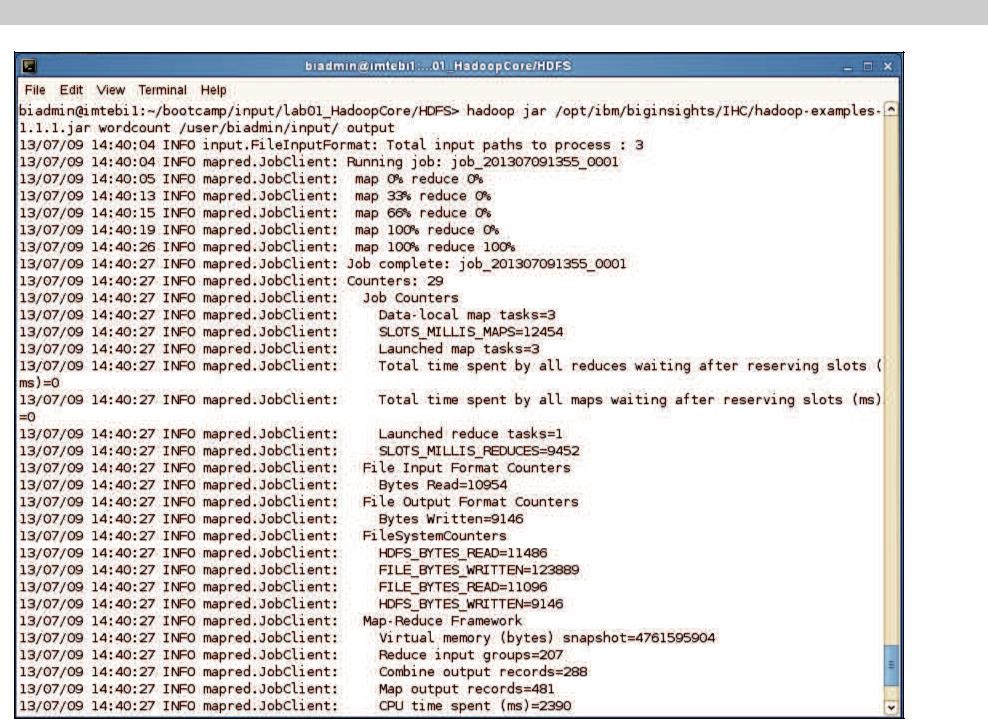
**hadoop fs -ls input**



**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**



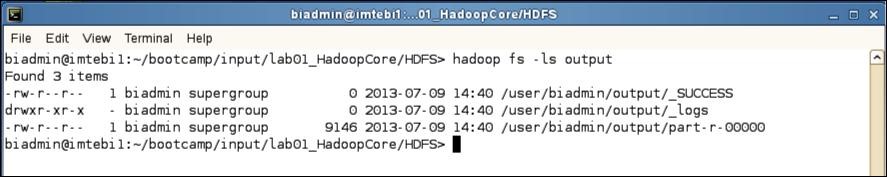
**WordCount MapReduce job running**

Step 4: Now review the output of step 3:

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



**hadoop fs -ls output**

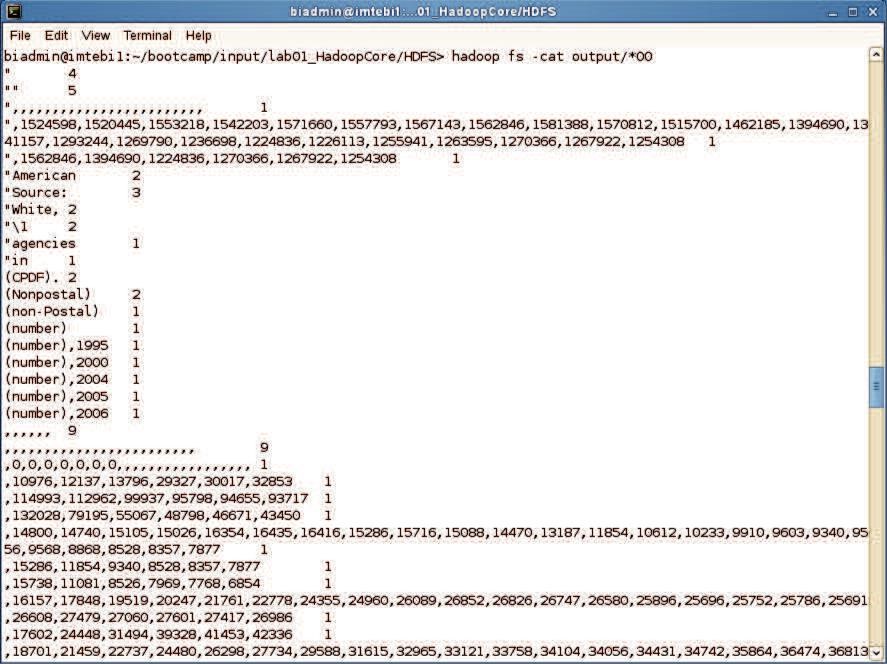


**MapReduce result files**

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**

[Text Wrapping Break]

**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.**

package www**.**ehadoopinfo**.**com**;**  
**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**);**  
 **}**  
 **}**  
 **}**  
**}**  
**program ends here**

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

package www**.**ehadoopinfo**.**com**;**  
  
**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**

package www**.**ehadoopinfo**.**com**;**  
  
**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**();**  
// M is an m-by-n matrix; N is an n-by-p matrix.  
 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. Executing the jar file using hadoop command and thus how fetching record from HDFS and storing output in HDFS.**

$ hadoop jar MatrixMultiply.jar www.ehadoopinfo.com.MatrixMultiply Matrix/\* result/

WARNING: Use "yarn jar" to launch YARN applications.

17/10/09 14:31:22 INFO impl.TimelineClientImpl: Timeline service address: <http://sandbox.hortonworks.com:8188/ws/v1/timeline/>

17/10/09 14:31:23 INFO client.RMProxy: Connecting to ResourceManager at sandbox.hortonworks.com/10.0.2.15:8050

17/10/09 14:31:23 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.

17/10/09 14:31:24 INFO input.FileInputFormat: Total input paths to process : 2

17/10/09 14:31:24 INFO mapreduce.JobSubmitter: number of splits:2

17/10/09 14:31:24 INFO mapreduce.JobSubmitter: Submitting tokens for job: job\_1507555978175\_0006

17/10/09 14:31:25 INFO impl.YarnClientImpl: Submitted application application\_1507555978175\_0006

17/10/09 14:31:25 INFO mapreduce.Job: The url to track the job: <http://sandbox.hortonworks.com:8088/proxy/application_1507555978175_0006/>

17/10/09 14:31:25 INFO mapreduce.Job: Running job: job\_1507555978175\_0006

17/10/09 14:31:35 INFO mapreduce.Job: Job job\_1507555978175\_0006 running in uber mode : false

17/10/09 14:31:35 INFO mapreduce.Job: map 0% reduce 0%

17/10/09 14:31:45 INFO mapreduce.Job: map 100% reduce 0%

17/10/09 14:31:53 INFO mapreduce.Job: map 100% reduce 100%

17/10/09 14:31:54 INFO mapreduce.Job: Job job\_1507555978175\_0006 completed successfully

17/10/09 14:31:55 INFO mapreduce.Job: Counters: 49

File System Counters

FILE: Number of bytes read=198

FILE: Number of bytes written=386063

FILE: Number of read operations=0

FILE: Number of large read operations=0

FILE: Number of write operations=0

HDFS: Number of bytes read=302

HDFS: Number of bytes written=36

HDFS: Number of read operations=9

HDFS: Number of large read operations=0

HDFS: Number of write operations=2

Job Counters

Launched map tasks=2

Launched reduce tasks=1

Data-local map tasks=2

Total time spent by all maps in occupied slots (ms)=15088

Total time spent by all reduces in occupied slots (ms)=6188

Total time spent by all map tasks (ms)=15088

Total time spent by all reduce tasks (ms)=6188

Total vcore-seconds taken by all map tasks=15088

Total vcore-seconds taken by all reduce tasks=6188

Total megabyte-seconds taken by all map tasks=3772000

Total megabyte-seconds taken by all reduce tasks=1547000

Map-Reduce Framework

Map input records=8

Map output records=16

Map output bytes=160

Map output materialized bytes=204

Input split bytes=238

Combine input records=0

Combine output records=0

Reduce input groups=4

Reduce shuffle bytes=204

Reduce input records=16

Reduce output records=4

Spilled Records=32

Shuffled Maps =2

Failed Shuffles=0

Merged Map outputs=2

GC time elapsed (ms)=196

CPU time spent (ms)=2720

Physical memory (bytes) snapshot=536309760

Virtual memory (bytes) snapshot=2506076160

Total committed heap usage (bytes)=360185856

Shuffle Errors

BAD\_ID=0

CONNECTION=0

IO\_ERROR=0

WRONG\_LENGTH=0

WRONG\_MAP=0

WRONG\_REDUCE=0

File Input Format Counters

Bytes Read=64

File Output Format Counters

Bytes Written=36

**Step 10. 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